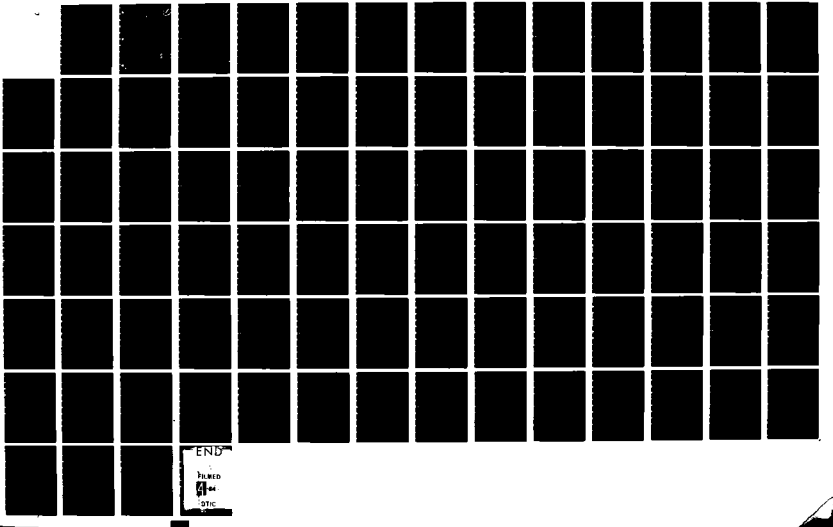
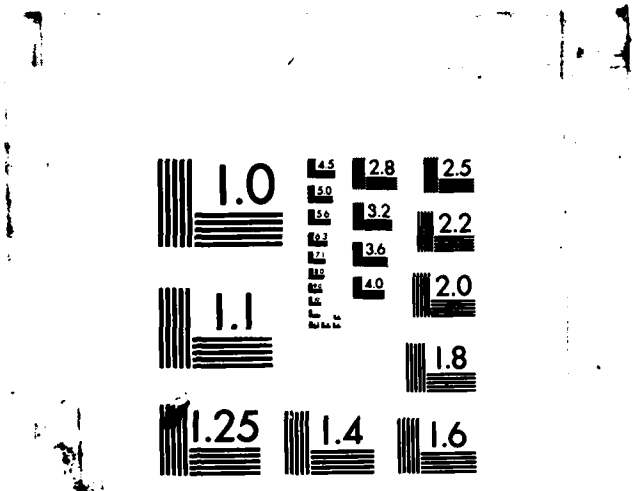
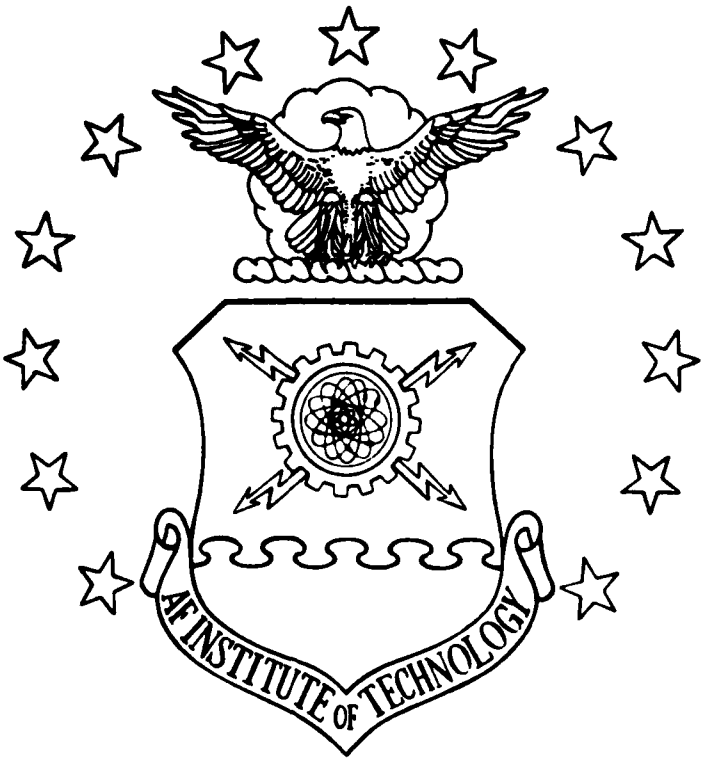
AD-A138 071 A NATURAL LANGUAGE INTERFACE FOR A PROLOG DATABASE(U) 1/1
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING R P WHITE 16 DEC 83 AFIT/GCS/EE/83D-22
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



A NATURAL LANGUAGE INTERFACE FOR A PROLOG DATABASE

THESIS

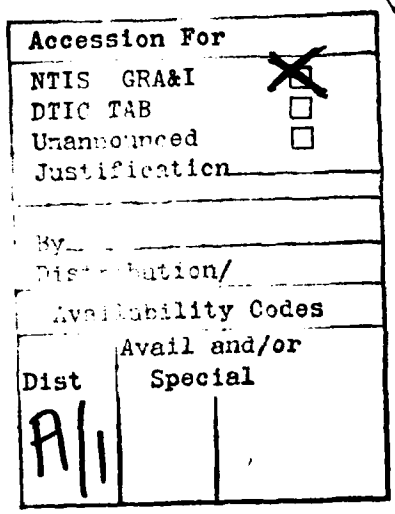
CAPTAIN ROGER P. WHITE, USAF

AFIT/GCS/EE/83D-22

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/83D-22



A NATURAL LANGUAGE INTERFACE FOR A PROLOG DATABASE

THESIS

CAPTAIN ROGER P. WHITE, USAF

AFIT/GCS/EE/83D-22

Approved for public release; distribution unlimited.

AFIT/GCS/EE/83D-22

A NATURAL LANGUAGE INTERFACE FOR A PROLOG DATABASE

THESIS

Presented to the Faculty of the School of Engineering

Air Force Institute of Technology

Air University

Submitted in partial fulfillment of the requirements for

the degree of Master of Science in Computer Science

by

Captain Roger P. White, USAF

Graduate Computer Science

16 December 1983

PREFACE

With the Japanese announcement of the Fifth Generation Project, and the implicit mandate to base the computational approach on logic programming, much attention has been drawn to the PROLOG programming language. Since my personal interests are centered in artificial intelligence, man-machine interfaces(MMI), and software engineering, the development of a natural language interface to PROLOG relational data provided an excellent vehicle to not only test some MMI theories but also to personally evaluate PROLOG and logic programming.

This thesis project presents the working kernel of a system where the database and linguistic components can be designed independently. Rationale is included to support: the use of PROLOG for such tasks; the use of the designed query language; and the use of such human-factors aids like: functions, aliases, and ellipsis.

I would especially like to thank Captain Robert W. Milne (advisor), who received his doctorate from the Department of Artificial Intelligence, University of Edinburgh. He provided some initial suggestions and the PROLOG environment, without which this project would have been impossible. Thanks are also due to my thesis committee members, Dr. Thomas Hartrum and Major Chuck Lillie. I also salute the developers of the software tools FINE and SCRIBE, who have made writing less painful for the rest of us. And finally I wish to acknowledge my gratitude to my wife and sons: Kathi, Nathan, Scott, and Mike for their support and good humor.

Table of Contents

List of Figures

List of Tables

ABSTRACT

A natural language (NL) interface to a relational database was developed with careful consideration given to the human factors needed to aid a novice user in accessing data. The entire system is written in DEC-10 PROLOG, with three distinguishing contributions:

(1) A simple grammar was developed to parse phrases like: 'Officers where rank is Captain and status is single and age is less than 32'. The semantic grammar was developed to be independent of the data domain so that both linguistic enhancements and database organization or normalization can be pursued independently and with minimum impact of the other component.

(2) The developed NL compiler accepts English phrases or commands as queries and translates them into PROLOG. The PROLOG system then does the database retrieval and display of information. But, since PROLOG does not allow global variables, some original techniques were developed to preserve common variable instantiations in a complex PROLOG object while being built by the parser.

(3) The human factors that contribute to the system are: a help file to aid user perception of the data, a simple grammar, an interactive view of all retrieved tuples, two forms of ellipsis, user-defined aliases, and limited use of quantifiers.

1. INTRODUCTION

The intent of this thesis project is to implement and report on a natural language interface to relational data. It was during the early months of research that the Japanese goals for 5th generation computing captured world attention [15, 16, 31, 34, 40]. Some of the goals include natural language processing, logic and very-high-level programming languages, high-level query languages, and knowledge representation which are also themes covered in this thesis effort. Also, along with the 5th generation came a mandate or manifesto for basing the entire computational approach or mechanism on logic programming [31]. A natural by product of this thesis will be the author's evaluation of logic programming for implementation of the current thesis objectives.

1.1 Background

Natural language interfaces to data and computing systems are both desirable and essential. Many top level military and government officers need vital and accurate information at their finger-tips. Society has already catapulted into the computer and information age. Even Time magazine, recently, featured the computer as "Man-of-the-Year" [21]. No one questions the ability of the computer to store and manipulate data. However, one of the main problems that impedes the shift towards the computer-aided society is that only experts and programmers have direct access to the machine.

In many cases, Generals, managers, and decision makers need critical information, without time delays. The problem is language. Currently, since only a programmer can "speak" to the machine, the time it takes for him to: understand the problem, design an algorithm, and correctly code a program, constitute a time wasting bottleneck. The programmer bottleneck, which can require days, or even weeks, would certainly be intolerable in a time of war or crisis. In most cases a dec' 'on maker does not want or need to program a computer; h lust needs information that can be accessed in a timely and nat way.

There is much current research being done in this area by professionals in natural languages, query languages, programming languages, database design and artificial intelligence. One of the newer approaches includes the use of the PROLOG system which unifies these research areas [9, 10, 11, 39]. It is natural in the PROLOG environment to view all data as relational and the access procedures or clauses as a query language. PROLOG can be viewed as either an extension of pure Lisp [40], or as an extension of a relational database query language. It has been used mostly in Europe on a wide variety of artificial intelligence tasks, and would be most suitable for the present thesis objectives.

1.2 Problem Statement

The objective of this research is to develop a natural (english) language interface to a PROLOG database, so that the

casual user of a computer can access information without the wasted time and effort associated with the "programmer bottleneck". The software system is named PROSEQ for: PROlog Structured English Query, language.

1.3 Scope and Assumptions

This thesis will contain sections that discuss the following five areas as sub-topics of research.

- Logic Programming: what it is, how Horn clauses are used in PROLOG, and the advantages of logic programming.
- PROLOG: why it was chosen for implementing the fifth generation of computers, its relationship to artificial intelligence programming, and how it supports the relational model of data.
- Query Languages: examine the query languages that exist for the relational model and design a suitable query language for this system that will serve both as a target language for the English language "front-end" and as an interface to the PROLOG query "back-end".
- Natural Language: design and write an English language parser that will handle simple and intermediate level English language queries for a relational model of data.
- Human Factors: the system should be easy to work with, simple to explain to a novice, and handle such things as ellipsis.

The system will be built with the intention of handling only that subset of English which one would use to interrogate a database . A modular approach will be used so that a simple working kernel is built first. The original system will only query the database and not make up-dates or modifications to the database. Finally, it is not the intention of the author to do

query optimization or the formatting of output; these can be accomplished in follow-on projects.

1.4 Summary of current Knowledge

Initial reference material for this thesis can found by researching material found in the following, which are presented here as a basic foundation for this problem domain.

- Artificial Intelligence - (natural language and knowledge representation) See the Handbook of Artificial Intelligence [2], Language as a Cognitive Process [42], AI Magazine, American Journal of Computational Linguistics [9, 41], ACM-SIGART [20, 35],and IEEE-COMPUTER [11].

- Logic programming - (PROLOG) See texts: Logic Programming [4], Programming in Prolog [5] and Algorithmic Program Debugging [32]. See also articles by Kowalski [23, 24], Manna [25],McCord [26],Pereira [28], Robinson [29],and Warren [38].

- Database - (the relational model and its query languages) See Codd [6, 7] and Date [12] also Astrahan [1],Dahl [10],Kim [22], Warren [39, 41],Waltz [36].

- Fifth generation computing - (Includes the integration of the first three areas) See [15, 16, 31, 34, 40, 43].

In current research implementations using PROLOG for a natural-language interface to a database the approach has been to translate from English [41] or Spanish [9] to logic and then query the database for relational data that has the same form as the logical interpretation of the query. The consequence of this approach is that the grammar writer and the database designer cannot operate

independent of each other. Facts or relations in the database have to conform to the linguistic assumptions of the logic grammar.

1.5 Approach and Resources

The approach in this project is to first build a simple kernel of features into a natural-language interface that will access any relational set of data where the grammar and database components can be designed independent of each other. After the kernel is functioning more complex linguistic structures can be added to the grammar as needed/desired. With this approach each database when loaded also supplies a description of the relational templates (or intentions) along with information for 'keying' the attribute fields of various relations together.

All of the software for this thesis was written in PROLOG on the DEC-10 of the USAF Avionics Laboratory. This thesis project focuses on three software components:

(1) A natural language compiler that handles a query subset of English
(2) An interactive system that allows the user access to the whole of PROLOG and the system interface at the same time
(3) A PROLOG relational database.

Since logic programming is modular, and parallel trends of thought are easy to abstract in Prolog, the various software components were developed concurrently with no interface debugging required! The vocabulary and database subcomponents of the system are encapsulated in separate files so that the natural

resulting modularity provides efficient use of the rest of the system across different domains of discourse.

1.6 Sequence of Presentation

The rest of the thesis is presented as follows. In chapter two the sub-problem topics of: Logic Programming, Natural language understanding, Query languages, and Human factors are discussed in more detail. Many issues and trade-offs could be discussed, but the intent of chapter two is to survey the sub-topic areas and establish potential implementation goals for the entire thesis project. At the end of each subsection of chapter two there is a rationale section that specifies the objectives or justifies the design or implementation decisions made for this project. These goals are intended as a summary of the subsection and as representing the author's choices in that subject area for implementation. In chapter three the software design and implementation are discussed based on the rationale set forth in chapter two. Chapter four reports the results of the actual implementation of the system and its software. Finally, in chapter five the conclusions are presented along with recommendations for further work and research.

2. DETAILED ANALYSIS OF THE PROBLEM DOMAIN

2.1 Logic Programming and PROLOG

The following sections present an introduction to the ideas behind logic programming and how it works. After this introduction, PROLOG itself is detailed for the convenience of the reader.

2.1.1 Computation with Horn Clauses

Logic programming is based on the subset of predicate logic that consists of Horn clauses as described below. For detailed formal descriptions of logic programming the reader is referred to Kowalski [24] and Clark [3]. Most of the following section is condensed from Eisinger [14] who is conducting research on methods for parallel execution of logic programs.

2.1.1.1 Syntax of Horn Clauses

We assume four disjoint sets of primitive symbols:

- A set of variable symbols denoted by u,v,w,x,y,z,...
- A set of constant symbols denoted by a,b,c,...
- A set of function symbols denoted by f,g,h,...
- A set of predicate symbols denoted by P,Q,R,...

Each predicate symbol and each function symbol has an associated integer which represents its arity. We define terms recursively. Each constant or variable is a term. If t1,t2,...tn are terms and f is a function symbol of arity $n > 0$, then f(t1,t2,...,tn) is also a term. If t1,t2,...,tn are terms and P is a predicate

symbol with arity n >= 0, then P(t1,t2,...,tn) is an atomic formula or atom. If A1,A2,...,An and B1,B2,...,Bm are atoms, and n >= 0, m>= 0, then B1,B2,...,Bm <-- A1,A2,...,An is a clause. And for m <= 1 the clause is called a Horn clause.

There are four types of Horn clauses:

- assertion : m = 1, n = 0 ; P(x) <--
- procedure : m = 1, n > 0 ; P(x) <-- Q(x),R(a,x)
- goal clause : m = 0, n > 0 ; <-- Q(x),R(a,x)
- null clause : m = 0, n = 0 ; <--

In PROLOG the asserted facts are generally stored in the database. The procedures are analogous to procedures or routines written by a user of any programming language, and the goal clause is that which is used to 'run' or query the interactive system at run time. In general the arrow will be dropped when refering to an assertion so the P(x) <-- will appear as P(x); and the null clause will appear as #, instead of the solo arrow.

2.1.1.2 Declarative Semantics of Horn Clauses

A set of Horn clauses can be regarded as a single formula in predicate logic by the following convention. Where a clause of the form,

B <-- A1,A2,...,An

represents the formula

(V x1,...xk)((A1 ... An) --> B),

where x1,...xk are all the variables in the clause and the symbols ∀, ~, ∧, ∨, -->, are the usual predicate logic symbols that respectively represent universal quantification, negation, conjunction, disjunction, and implication. By definition, any variable in a clause is universally quantified. The same variables that appear in different clauses are distinct.

2.1.1.3 Procedural Semantics of Horn Clauses

In addition to the static declarative semantics, a set of Horn clauses may be interpreted dynamically. Assertions and procedure clauses are regarded as procedure definitions, goal clauses as procedure calls and the empty clause as a halt statement. Unification corresponds to parameter passing and the returning of results.

Program execution is non-deterministic. To achieve deterministic behavior an order has to be defined for the search space, (or in Prolog the database). In Prolog this is accomplished in a simple and efficient way: the atom selected in a goal clause is always the leftmost atom. The selected atom is resolved against the first matching procedure clause in the same order that the clauses are written (stored). The clause chosen for expansion is always the clause most recently generated. If a failure occurs, the process backtracks to the last point where a clause was selected and the next possible match is tried. Thus in Prolog the search tree is traversed in a left to right depth-first strategy.

2.1.2 PROLOG

PROLOG is both the implementation language and the intermediate query language for this thesis. Since Prolog is not yet widely known in the U.S. many of the basic concepts and the syntax are introduced here to make the thesis self-sufficient. Prolog was developed from Horn clause theory, largely the work of Robinson [29] and Kowalski [24] and the language itself was implemented on the DEC-10 by Warren et al [37]. Prolog has been chosen by many programmers for such symbolic computation tasks as the following:

* relational databases
* abstract problem solving
* architectural design
* ai - expert systems
* mathematical logic
* natural language
* symbolic equation solving
* 5th generation computing

2.1.3 PROLOG Syntax

The basic Prolog syntax is quite simple; where parenthesis are used to enclose (object lists), square brackets are used to enclose [lists], ":-" means "if", commas are used for conjunctions, and all facts and rules must end with a "." stop character. Also, variables in Prolog begin with a capital letter, all other atoms, whether in lists or relations must begin with a lower-case letter. There are two basic forms to remember. First, there are facts of the form:

```
relationship(object1,object2,...,objectn).

example: gave(roger,flower,kathi).

meaning: "Roger gave a flower to Kathi."
```

Second, there are rules of the form:

```
<goal> :- <sub-goal1>,<sub-goal2>,...,<sub-goaln>.

example: aunt(X,Y) :- female(X),
                      sibling(X,Z),
                      parent(Z,Y).

meaning: "X is the aunt of Y
              if X is female, and
                 X is a sibling of Z, and
                 Z is the parent of Y.
```

Prolog begins by searching the data base of facts and rules, and instantiates the variables to the first successful match. When each sub-goal has been satisfied then the main goal will be successful.

2.1.4 PROLOG Concepts

In Prolog there are five main concepts that govern the use and understanding of the language. They are: (1) objects, (2)relationships, (3) facts, (4) rules, and (5) questions. Relationships behave much the same way as relations in a relational database. Each fact is a relation with an initial functor followed by its related objects. The fact can be simple:

```
likes(scott,apples).
```

or more complex:

```
owns(roger,book(prolog_programming,
                authors(clocksin,mellish))).
```

Rules are the basis for Prolog programming, and are covered in the next section. Since Prolog is a conversational or interactive programming language, the programmer "asks" questions of the

Prolog system by presenting 'goal' clauses to the interpreter, which then searches the facts and rules that are already stored in the database. For example; the "likes(scott,apples)." rule, can be used four ways. Recall that, atoms beginning with a capital letter are variables.

```
query: ?- likes(scott,apples).
meaning: "Does Scott like apples?"
response: yes

query: ?- likes(Who,apples).
meaning: "Who likes apples?"
response: Who = scott

query: ?- likes(scott,What).
meaning: "What does Scott like?"
response: What = apples

query: ?- likes(X,Y).
meaning: "X likes Y?"
response: X = scott   Y = apples
```

In the last three examples, the variables: Who, What, X, and Y can be instantiated to different answers with repeated use of the query if there are more "likes(name,object)" rules in the database. A 'findall' routine can retrieve all objects that satisfy a given query. This then serves as the beginning of a relational database system.

2.1.5 PROLOG Programming

Programming in PROLOG consists of :

- Asserting some facts about objects and their relationships. In this project the facts will be the tuple extentions of a relational database.

Defining rules or procedures for manipulating objects and relationships. In this case the NL compiler was written as a set of grammar and semantic rules.

- Asking questions about the objects and relationships. For this project the interactive questions are presented in English to query the database.

2.1.6 Rationale

Since Prolog programming embodies the concepts of object and symbolic processing for relational data and can easily implement parsers by simply specifying them in a high level language, PROLOG is logically (pun intended) the best choice for this project. In has already been demonstrated by other researchers that PROLOG is well suited for natural language and database research, but the approach here will be to develop a system where both the natural language and the database components are developed independently for effective use in the same system.

2.2 Natural Language Understanding

2.2.1 Introduction

Much of the basic outline for this section of the thesis was derived from William Gevarter's [18] NASA memorandum on natural language processing. One of the major goals of Artificial Intelligence(AI) research has been the development of natural language understanding; or raising the man-machine interface from the programmer level to the common man level. Many of the first systems built where used to retrieve information from static files or access well defined databases. Current research seeks to increase the levels of linguistic complexity that a machine can

efficiently understand as well as increasing the complexity of knowledge represented and infering facts that are not explicitly stated.

2.2.2 Applications for Natural Language Interfaces

There are many applications for computer natural language understanding systems.

Table 2-1: Some Applications of Natural Language Processing.

Information Acquisition or Transformation	**Language Generation**
Machine Translation	Text Generation
Document or Text Understanding	Speech Output
Automatic Paraphrasing	Writing Aids
Knowledge Acquisition and Compilation	Grammar Checking
Information Access	**Interacting with Machines**
Information Retrieval	Control of Complex Machines
Question Answering Ssystems	
Computer-Aided Instruction	
Interaction with Intelligent Programs	
Expert Systems Interfaces	
Discourse	Decision Support Systems
Speech Understanding	Explanation Modules
Story Understanding	for Computer Actions

Some are listed in Table 2-1. With the maturing of the AI fields of natural language and computational linguistics the author believes that eventually most machines will have some kind of natural interface capability, allowing the common man to access data and work with it effectively without being a programmer. In fact this is one of the main goals of fifth generation computing.

2.2.3 Approaches

Natural Language Processing (NLP) systems utilize both linguistic and domain knowledge to interpret the input. Domain knowledge or knowledge of the subject area of discourse is so important to understanding that it is usual to classify the various NLP systems by the way this knowledge is represented. Hendrix and Sacerdoti [19] classify systems as A,B or C, where Type A is the simplest and least costly.

Briefly then, the type A systems were the first ad hoc approaches to NL understanding. Usually there were stored facts about a limited domain and the input was scanned for predefined key words, or patterns with known relationships among its objects. Using this simple approach, while ignoring the complexities of language, early systems were able to achieve impressive results. Type B systems encoded knowledge about the domain in frame or network representations that allowed the system to understand objects and relationships in the context of the over all knowledge and in terms of the expectation for the situation context. Schank's work with SAM is a good example of this approach [30]. Type B systems use explicit world models whereas the type C systems include information about the goals or beliefs of the intelligent entities; these advanced systems attempt to include in their knowledge base information about the intentions of the participants. Again see Schank [30].

Of course this is just one way to classify NLP systems, in the

next section a good case can be made for classifying systems by the linguistic methods used to to understand natural language

2.2.4 Grammars and the Parsing Problem

Actual linguistic knowledge of sentence parts and their relationship to the domain of discourse is essential for systems that are more complex than the keyword-template matching methods. The big problem is how to relate linguistic structure to the knowledge that is accessed and how to represent the knowledge. To address part of the problem the computational linguistics community studies syntax,semantics, and pragmatics. Syntax is the study of symbolic structure; semantics, the study of meaning; and pragmatics the study of the use of language in context. Table 2-2 lists some of the more popular grammars used for parsing natural language.

Table 2-2: Grammars for Natural Language Parsing

1. Phrase Structured Grammars(context-free)
2. Transformational Grammars
3. Case Grammars
4. Semantic Grammars
5. Extended Grammars (ATNs and Charts)
6. Metamorphis and Definite Clause Grammars

Brief descriptions of these grammars can be found in Barr [2] and Gevarter [18] with more extensive bibliographies and formal treatment found in Colmerauer [8], Gazdar [17] and Winograd [42]. Having decided to use Prolog for this Natural language/ Database project; the decision to use a semantic grammar embedded

in a Definite Clause Grammar(DCG) formalism was a natural consequence. The definitions and justification are presented in the following sections.

2.2.5 Semantic Grammars

A semantic grammar partially or fully eliminates the traditional linguistic elements of NL grammar and replaces them with concepts, abstractions, or classes that have meaning in a particular domain or context. For example, the grammar (see figure 3-2) for this project is defined in terms of: relations, conditions, attributes, and operators, rather than: nouns, verbs, adjectives, prepositions, and their corresponding phrases. This approach has a two fold advantage. First, the problem can be abstracted in terms of its semantics, in this case a database. And, second, all of the computational overhead needed to lift semantic information from a traditional parse tree is eliminated. All that is needed then for this project are semantic components of the compiler that build the implicit selects and joins into a executable PROLOG object. The great advantage of a semantic grammar is that much of the usual semmantic processing can be embedded into the syntactic element of the compiler.

2.2.6 Definite Clause Grammars

The Definite Clause Grammar is a formalism originally described by Colmerauer [8], which is a first-order predicate logic generalization of context-free grammars. The following is quoted from Pereira and Warren [28].

> DCGs are a natural extension of CFGs. As such, DCGs inherit the properties which make CFGs so important for language theory: the possible forms for the sentence of a language are described in a clear and modular way; it is possible to represent the recursive embedding of phrases which is characteristic of almost all interesting languages; there is an established body of results on the CFGs which is very useful in designing parsing algorithms... Now if a context free grammar is expressed in definite clauses according to the Colmerauer-Kowalski method, and executed as a PROLOG program, the program behaves as an efficient top-down parser for the language... This formalism, which we call a definite clause grammar is a normal form of what Colmerauer calls a metamorphosis grammar.

Pereira and Warren continue with the advantages of a definite clause grammar by describing three important extensions of the DCG that are not found in the context-free grammar formalism. First, a DCG can provide for context-dependency; Second, a DCG can allow arbitrary tree structures to be built...which are not constrained by the recursive structure of the grammar; and Third, DCGs allow extra conditions and parameters to be included in the grammar rules; these added conditions make the course of parsing depend on auxiliary computations and provide for simple direct implementation of semantic routines of the compiler, to an unlimited extent. This third advantage is the ideal foundation for attribute grammar implementation and was used extensively in this thesis.

2.2.7 Knowledge Representation

There are many ways that knowledge can be represented and manipulated. But, with the choice of PROLOG, the natural choice is the first-order predicate calculus, even though other methods

could be implemented in PROLOG. This section lists the many advantages of using logic with the following quoted from Barr and Feigenbaum [2].

1. Logic often seems a natural way to express certain notions. As Mccarthy and Filman pointed out, the expression of a problem in logic often corresponds to the intuitive understanding of the domain. Green also indicated that a logical representation was easier to reformulate; thus, experimentation is made easier.

2. Logic is precise. There are standard methods of determining the meaning of an expression in the logical formalism. Hayes presents a complete discussion on this same issue and argues for the advantages of logic over other representation systems.

3. Logic is flexible. Since logic makes no commitment to the kinds of processes that will actually make deductions, a particular fact can be represented in a single way, without having to consider its possible use.

4. Logic is modular. Logical assertions can be entered in a database independently of each other; knowledge can grow incrementally, as new facts are discovered and added. In other representational systems, the addition of a new fact might sometimes adversely affect the kinds of deductions that can be made.

Also, databases are a particularly attractive application area for logic programs. The following list comes from a list compiled by Veronica Dahl [11].

1. Facts and rules can coexist in the description of any relationship

2. Recursive definitions are allowed.

3. Multiple answers to the same query are possible. This feature is usually referred to as nondeterminism.

4. There is no input/output role distinction for a predication's arguments. Thus, the same PROLOG description of a relationship can be used in a combination of ways, depending on the placement of variables and constants in the argument fields.

5. Inferencing takes place automatically.

These features have important implications for database applications. Because facts and rules can be used, no separate deductive compoment is needed - unlike most conventional database systems. Moreover, since recursive rules and alternative answers are allowed, the user can arrive at very clear, concise, and nonredundant descriptions of the data at hand. Because of the non-distinction between input and output, any argument or combination of arguments can be chosen for retrieval, whereas conventional databases must name and control different paths to make this possible. Finally, the fact that answers are automatically extracted using the data description by a user-invisible inference procedure results in a great degree of data independence; not only is the user allowed to represent data in higher level, human-oriented terms rather than in terms of bits, pointers, arrays, etc., but he is also spared the effort of describing the operations used to retrieve it. These operations are implicit in the PROLOG mechanisms, which give an operational meaning to the purely descriptive facts and rules used.

2.2.8 Rationale

The definite clause grammar or metamorphosis grammar formalism is computationaly equivalent to a predicate logic generalization of a context-free grammar. The approach here will be to design a semantic grammar than is generalized over the relational model of data rather than over a very narrow single application domain. In this way the same grammar and parser can used on interchangeable sets of data. This modular approach gives the grammar writer and the database implementer independence explained in chapter three.

2.3 Artificial Intelligence and Database

2.3.1 Introduction

During the past several years the disciplines of artificial intelligence and database have been moving closer together to solve common problems. Some of them being : how to represent data or knowledge , natural language interfaces, conceptual modeling, and logic and deduction. Natural language research aims to increase linguistic coverage, and there has been much progress in this endevor. However, NL systems are far from guaranteeing that all input is properly understood. There are two interim solutions. The system can be designed to be conversational where the user is interrogated until the query is clarified. Or, the second approach is to constrain the NL input so the question is sure to be understood. This is the approach adopted for this thesis. In the following sections the human factors for query languages will be discussed followed by the rationale for the design of the PROSEQ query language. Also, with the choice of PROLOG as the implementation vehicle, the relational data model is assumed.

2.3.2 Query Languages and Human Factors

A recent U.S. Army technical report [13] on the design recommendations for query languages has stated that; "the existing human factors literature on query languages is both sparse and scattered...and most of the research done on human factors in query languages has been of limited scope". This

section then will be based on the findings of the Army's report and the author's personal observations while using other query languages.

The stated purpose or requirement for the Army's report [13] was to improve the design of query languages by making them simpler to use, easier to learn and less prone to user error. The following five paragraphs summarize some of the findings and recommendations for query language design.

2.3.2.1 User's Perception of the Database

The user's view of the data in a formatted database has a fundamental impact on the way he conceives and formulates queries. Therefore, the organization of the database should be in accord with what is perceived to be natural by its users. And, the user's perception of the database should be sufficiently structured so as to enable rapid, natural identification of those parts in which the user is interested.

2.3.2.2 Natural Query Languages

The arguments for and against natural query language may be summarized as follows:

> Detractors feel that (1) natural language is too ambiguous to serve as a computer language and (2) when learning to use formal language, one also learns to formalize the process of problem solving. In other words, using a formal language involves a change in the way one thinks as well as a chabge in the syntax and vocabulary. On the other hand, supporters of natural query languages contend that (1) citing examples of NL ambiguities does not constitute proof that English cannot work as a computer language and (2) natural query languages are not

> intended to lighten the burden of having to think. Rather, their advantage lies in eliminating the need to remember a host of notational devices which are irrelevant to the problem and which detract from the user's ability to concentrate on the problem per se. In conclusion, the desirability of using English as a computer language has been debated heatedly...

However, this author feels that natural constraints can be designed into NL query systems that eliminate the ambiguity problem and still retain powerful compact query expression with broad linguistic coverage.

2.3.2.3 Formal Query Languages

Formal query languages, characterized by highly structured syntax, were designed for ease of learning and use. SEQUEL is a good example of such a language where :

```
SELECT      name
FROM        grades
WHERE       class = 'ma531'
AND         grade = 'B+'
```

means "Find the names of B+ students in math 531. One of the objectives of this project is to incorporate the essential elements of a formal query language into a less structured natural query language, while retaining the same computational power. More details are given in the rationale paragraph.

2.3.2.4 Language Aids or Options

Aside from the strict definition of a query language, other 'non-essential' features can improve the human interaction with a system. Some of these are: abstraction capabilities, ellipsis,

and user or system defined functions. MINIMUM, MAXIMUM, and AVERAGE are examples of useful functions. The use of ellipsis can speed-up the query process by eliminating the need to duplicate input that can easily be infered from the context of the dialogue. And finally, abstraction capability, in the language, allows one to re-name long strings with shorter ones. A long, involved query, that is used say at the end of each month, could be invoked with a simple, 'EOM-REPORT'.

2.3.2.5 Quantifiers

Thomas [33] has reported that the average user has great difficulty in using quantifiers correctly when formulating queries. This difficulty is not unique to query languages. Instead, people in diverse situations appear to have difficulty in using quantifiers in the way of logicians.

2.3.3 Design Rationale for the PROSEQ query language

Having examined the above mentioned human factors and given the previous decisions on PROLOG, definite-clause grammars, semantic grammars, etc., the following criteria were set forth for the design.

- User perception of the database will be aided by a help file that contains intension templates of the relations in the database along with information on how the relations are 'keyed'. Also, the query grammar and typical example queries will be displayed.

- The initial query language will be kept lean and simple. The user can first specify, in a natural way, the relations needed for the query and then second, list the conditions to be satisfied by the selected set of relations.

- The system will be interactive to allow the user to see all of the data used and the results of the selects and joins. In this way the user can do immediate interactive error checking. A final project function can be used to obtain the desired format for the query output.

- The system options or aids will include : a help file, two kinds of ellipsis, and abstraction capability.

- The only quantifiers that will be understood by the system will be 'all' or 'none'. The system will respond with all the data sets that satisfy a query or with none (meaning an error or empty set).

- The vocabulary will be limited to names of attributes in the database and the words in each domain, however provision will be made for synonmyms and abbreviations.

3. PROJECT DESIGN AND IMPLEMENTATION

In this chapter both the features and system design are described in detail. The first section on features summarizes the goals as discussed through out chapter two. The rest of the chapter is dedicated to a description of each conceptual part of the PROSEQ system.

3.1 Features

The desired features for this system appear in

Table 3-1: Initial Implemented features of the PROSEQ System

1. A Semantic grammar.
2. Handles multiple conditions across multiple relations.
3. Operators include: and,or,not, equal,less than, greater than.
4. Elipsis: two kinds
5. A Help file
6. Direct access to the underlying PROLOG system(for degugging)
7. Modularity & Portability
8. User defined phrases
9. Can use synonyms and abbreviations

Table 3-1. Each item will be discussed in more detail in this and the next chapter.

3.2 Approach

The initial desire of the author was to have a thesis project that would integrate the fields of AI, NL, DB, and human factors, the fact that Prolog and the 5th Generation Projects appeared on the scene at the same time as initial research was begun, was

coincidental but very much welcome. The initial idea for an approach was discovered while reading the section on Definite Clause Grammars (DCG) of the USER'S GUIDE to DECsystem-10 PROLOG [27], which appears as Appendix I in this thesis. In the DCG section a small interpreter is defined, see Figure 3-1, which parses an arithmetic expression (made up of digits and operators) and computes its value.

Figure 3-1: A Parser and Semantics for Arithmetic Expressions

```
expr(Z) --> term(X), "*", expr(Y), {Z is X * Y}.
expr(Z) --> term(X), "/", expr(Y), {Z is X / Y}.
expr(X) --> term(X).

term(Z) --> number(X), "+", term(Y), {Z is X + Y}.
term(Z) --> number(X), "-", term(Y), {Z is X - Y}.
term(Z) --> number(Z).

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {"0"=<C, C=<"9", X is C - "0"}.
```

Of particular note in Figure 3-1 is the clean representation and the separation of the syntactic and semantic elements of the specification. The instantiated variables and the instructions appearing between the curly brackets '{' and '}' are the semantic elements of the interpreter; everything else is the syntactic parser. In logic programming a compiler or interpreter is its own specification [40]. This then became the obvious, elegant approach for the natural language element of the project.

3.3 System Overview

The Figure 3-2 is depicted here to show how the entire system is organized. This is intended as a conceptual view that

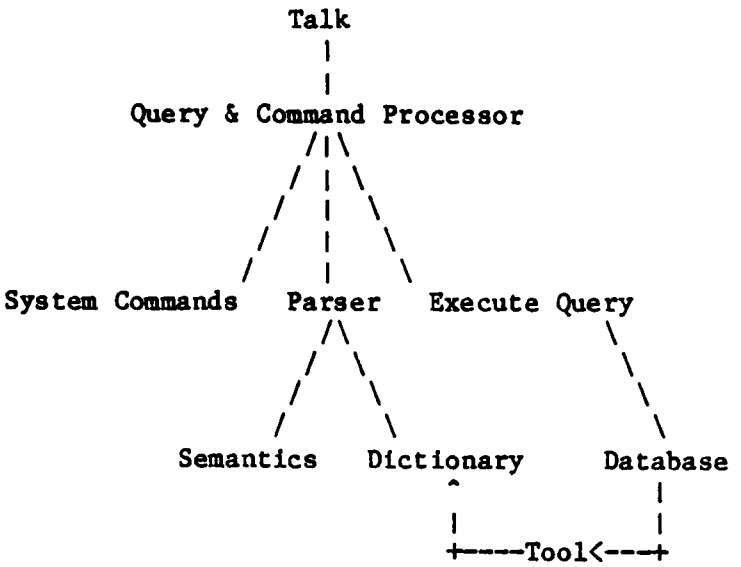


Figure 3-2: The Components of PROSEQ

represents the relationships which exist between the independent PROLOG clauses that constitute the functional elements of the system. The details for each node in the system tree and how the separate entities interact is described below. The system should be viewed as a natural language compiler that generates Prolog objects which are then executed against the relational database.

3.3.1 Talk

Abstracting the top level of the PROSEQ system, one would say: " I can 'talk' to PROSEQ if the system will repeatedly request input, that is either a query or a command, and respond to such, until told to stop". And since Prolog is ideal for abstracting in this way the actual code would look like :

```
talk :- repeat,
        write('Enter Query or Command'),
        read_in(Sentence),
        respond_to(Sentence),
        Sentence = [stop|_].
```

The Prolog code for the "read_in" clause is the same as that found on pages 86-88 of the Clocksin and Mellish book [5] on Prolog programming. The rest of the code, except for a few small utilities, is original and is found in Appendix IV.

3.3.2 Query and Command Processor

The previous "respond_to" clause is actually called "quecomp" in the implemented code. The "quecomp" clause is also synonomous with the the start symbol of the grammar used to parse either a command or a query. This means that there is no actual code for "quecomp" as a processor. All processing (changing natural language to Prolog language) is accomplished in the semantic routines of the parser. "Query and Command Processor" can be viewed as a compiler, where "quecomp" is the start symbol of the grammar. During parsing, appropriate semantic routines are called on to build Prolog structures or objects, then the rest of the semantic work is accomplished by the Prolog system with simple, "call(object)" procedure. This is one of the principle contributions of this thesis. Since Prolog data is relational, and Prolog can be viewed as a powerful query language by itself; then the natural language element of the system should generate Prolog queries, which are then executed with the "call(object)" command.

3.3.3 System Commands

Initially the only PROSEQ system command was 'stop'. But while implementing this feature the author discovered the second contribution of this thesis; that all of the Prolog system can still be accessed easily without leaving (and then having to return to) the PROSEQ environment. This means that PROSEQ will accept both natural language and Prolog as inputs, which can be very useful during debugging sessions, where it is very annoying to have to abort a run, get out of PROSEQ, do some things in PROLOG, return to PROSEQ, and start the run over again. Table 3-2 lists just a few of the commands that users may find useful.

Table 3-2: Some PROSEQ System Commands

stop.....to terminate 'talk' and return to Prolog

help.....to introduce the novice to the system

listing.....to peek at resident clauses

debug.....to turn the debugger on

spy.....to spy on certain clauses for debugging

The current implementation of PROSEQ only parses simple clauses with one or two parameters, but this can easily be extended to cover all of Prolog.

3.3.4 Parser

The grammar is depicted in Figure 3-3, where S is the start symbol and '|' is used for alternation. The NON-TERMINALS are in all capital letters, while [terminal-symbols] are in lower case

characters enclosed in square brackets. Currently the grammar for the parser looks like this :

```
S ::= RELATIONS [where] CONDITIONS [.]
S ::= [and] CONDITIONS [.]
S ::= [where] CONDITIONS [.]

RELATIONS ::= RELATION
RELATIONS ::= RELATION [and] RELATIONS

RELATION  ::= [officers] | [grades] | [students] | ...
              (any word that is the name of a relation)

CONDITIONS ::= CONDITION
CONDITIONS ::= CONDITION [and] CONDITIONS
CONDITIONS ::= CONDITION [or ] CONDITIONS

CONDITION ::= ATTRIBUTE OPERATOR VOCABULARY

ATTRIBUTE ::= [name] | [age] | [class] | [code] | ...
              (any word that names an attribute domain)

OPERATOR  ::= [is] | [is not] | [is less than] |
              [is greater than] | [equals] |
              [=] | [<] | [>] | [~]

VOCABULARY ::= [roger] | [married] | [ma531] | ...
               (any word that appears in the attribute
                     fields of the relational database)
```

Figure 3-3: The Grammar for PROSEQ

Some sample sentences that can be parsed by this grammar would be:

```
Officers where rank is captain and status is married.
Students and grades where sex is female and class is cs565.
Officers and students where age is greater than 21 and
      age is less than 30 and status is single.
```

An added feature of the grammar is that ellipsis may be used, if subsequent queries use the same relational context as the current

```
|:   talk.
Enter Query or Command
|: officers where rank is captain.

      officer(roger,30,male,captain,married,cs)
      officer(mike,29,male,captain,married,cs)
      officer(mary,33,female,captain,divorced,log)
      officer(suesan,24,female,captain,single,log)
      officer(tom,28,male,captain,married,ee)
      officer(kathi,26,female,captain,single,ee)

Enter Query or Command
|: and sex is female.

      officer(mary,33,female,captain,divorced,log)
      officer(suesan,24,female,captain,single,log)
      officer(kathi,26,female,captain,single,ee)

Enter Query or Command
|: and major is ee.

      officer(kathi,26,female,captain,single,ee)

Enter Query or Command
|: where major is ee.

      officer(linda,23,female,lt,single,ee)
      officer(david,45,male,major,single,ee)
      officer(tom,28,male,captain,married,ee)
      officer(kathi,26,female,captain,single,ee)

Enter Query or Command
|: where sex is female and status is single.

      officer(linda,23,female,lt,single,ee)
      officer(suesan,24,female,captain,single,log)
      officer(martha,22,female,lt,single,log)
      officer(kathi,26,female,captain,single,ee)

Enter Query or Command
|: stop.
```

Figure 3-4: Samples of ellipsis, using 'and' and 'where'

query. There are two cases: (1) if the user wants to further restrict the meaning of the last query processed then use, "and <conditions>"; (2) if the user wants to use the same relations but supply all new conditions then he would use, "where <conditions>". See Figure 3-4 as a sample session using these features. One caution: the user must have used a full phrase previous to the use of the ellipsis features so that the relations that are to be used have already been pushed onto the context stack.

3.3.5 Semantics

While the PROSEQ parser is recognizing the simple natural language strings, the semantic routines transform them into a PROLOG queries. Since pure PROLOG has no global variables, a stack is used to store intermediate structures; popping unfinished portions and then pushing additional information until the parse is finished and the structure is completed. The semantic routines are invoked in three phases. In the first phase the relational tuple-templates are concatenated together until the parser reaches the word "where". At this point all of the tuple-templates have been collected so the phase two semantic routine is called. In phase two the templates are "keyed" to each other in the same way and for the same reason that the relations are keyed in a relational database. In PROLOG this is done by instantiating "keyed" attributes with the same variable. The phase three semantic routines continually concatenate "conditions" which further restrict the meaning of the relational

templates. Also, each time a condition is translated the attribute variable must be instantiated with not only the correct relational template, but also the correct field within the designated template. Perhaps an example would be more illuminating. In Figure 3-5 the preceeding steps are re-created for the specified example. During the compiling process the intermediate PROLOG structure is stored on a stack while the parser or syntactic work is being performed. When ever semantic work is required the stack is popped, the appropriate concatenations and variable instantiations are performed, and then the new PROLOG structure is pushed back on the stack, and parsing resumes. The method used by the semantic routines is essential for preserving the common variables (or keys) of the query, since PROLOG itself makes no provisions for global variables.

3.3.6 Dictionary

The dictionary is nothing more or less than all of the words that appear in the domains for each attribute of all the relations for users specified database. In the current implementation the dictionary is implemented with one logical rule:

```
vocab(Word) --> [Word].
```

which means that the parser and semantic routines will except any word (nonsensical or misspelled) that is supplied and does not detect an error until the search is executed against the

[Step 1] The query as it appears when first entered by the user.

```
|:Officers and grades where rank is captain and status is married
|: and grade is A.
```

[Step 2] The query as a string prepared for the parser.

```
[officers,and,grades,where,rank,is,captain,and,
status,is,married,and,grade,is,a,.]
```

[Step 3] During phase one the relational templates are collected with PROLOG supplying a unique variable for each field.

```
(grade(_1,_2,_3),officer(_4,_5,_6,_7,_8,_9))
```

[Step 4] The phase two semantic routine establishes the keys as defined by the database for these specific relations. This is done by re-instantiating key fields with the same variable.

```
(grade(_1,_2,_3),officer(_1,_4,_5,_6,_7,_8))
```

[Step 5]The phase three semantic routines continue to concatenate conditions and key them to the correct fields at the same time.

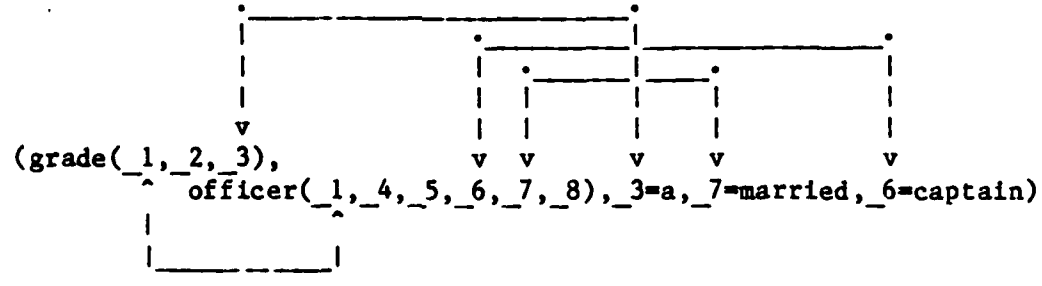


Figure 3-5: The Three Phases of the Semantic Routines

database. For example, the nonsense words 'tuesday' and 'divorced' in a phrase like:

officers where rank is tuesday and age is greater than divorced.

are not detected as an error until the database is searched and 'tuesday' and 'divorced' are not found in the domains for rank and age respectively. However, this approach has three advantages; first, no actual dictionary with hundreds or thousands of rules like:

```
vocab(male) --> [male].
vocab(female) --> [female].
vocab(captain) --> [captain].
        .
        .
        .
```

need be implemented, second, the problem of updating the vocabulary dictionary when ever the database changes goes away, and third, there is no 'type-checking' overhead. Also , the assumption is made that this kind of 'type-checking' can be done away with since that user is assumed to be intelligent and friendly. The one disadvantage of not type-checking is that un-intentional errors could be made by anyone. This problem is over come with a message from the system that states something like, " the word 'divorced' does not appear in the AGE domain". The user can then examine the word for misspelling or examine the database or help files to ensure that he has the correct understanding of the relations and domains as implemented for the current application.

3.3.7 Execute Query

The simplest part of the system is the execution of the query. The PROLOG system does all of the searching internally and then returns a list of objects that are valid responses.

After the compiler has finished building the PROLOG query object, the use of a "call(object)" command would only retrieve the first instance of valid answers for the query. Since all valid answers are desired, a simple 'findall' routine is invoked instead, and all the correct answers appear on the users screen in tuple form just as they are stored in the database. The code for 'findall' can be found near the end of the system code, which is included as Appendix IV.

3.3.8 Database

The database used to test this system contains the 400 rules (or facts) and three relations as presented in Appendix III. The three relational templates are:

```
officer(Name,Age,Sex,Rank,Status,Ed-code).
grade(Name,Class,Grade).
instructor(Class,Teacher-Name).
```

where 'status' is marital status and 'ed-code' is the officer's academic major or school to which s(he) is assigned.

The system was designed to access any number of relations, which can be restricted by any number of conditions. The complexity of how all the relations are keyed and relate to each other is isolated in the 'rlxmatch' (relational-cross-match)

routine. The implementation is both generic and modular. To achieve this, it is necessary to load a table or matrix which tells 'rlxmatch' which relations are keyed to each other and which fields are involved. In this way the semantics of the routine can remain unchanged from one database to the the next.

3.3.9 Tool

The purpose of 'Tool' is to save the implementor a lot of work when a dictionary needs to be implemented. Tool reads a the database in its raw relational form and automatically produces the dictionary for that specific database. In this effort 'tool' is not used as explained above.

4. RESULTS

The results for this thesis project are presented here in two parts; first, a discussion of the natural language or compiler element of the project, and second, a discussion of the human factors involved. The entire project is viewed as being a success with several follow-on thesis topics being proposed in the recommendation section of chapter five.

4.1 The Natural language Compiler

All of the objectives detailed in table 3-1 were implemented succesfully. The following representative queries were all compiled correctly and retrieved the specified information from the database.

```
Officers where rank is captain.
Officers where status is single and sex is female.
and major is ee.
where major is ee.
and age is greater than 34.
Officers and grades where status is divorced and grade is c.
where name is alice and grade is not a.
where name is roger.
where class is ma520.
Officers and grades and instructors where teacher is milne
and grade is b+ and major is ee.
Grades where grade is b or grade is c.
```

A complete demonstration is presented in Appendix II, where all implemented features are correctlly exercised. The system also allows direct access to the underlying PROLOG environment. This feature was very helpful during debugging, since it allowed the author to look at and change internal stacks of PROSEQ without leaving the home environment to make the changes. The system

easily accomodates the use of synonyms and abbreviations, which are implemented as extra grammar rules for a common terminal symbol. The semantics remain the same.

4.2 Human Factors

This itemized list represents those elements which make the system either more tractable for the implementor or more user-friendly for the novice user.

- 1. The grammar is powerful enough to do relational queries, yet simple enough to be memorized. One need only remember that relations preceed conditions and that ellipsis may be used once the relations have been specified.

- 2. A Help file can be called at any time to display the relations currently in use, how they are keyed, sample queries, and a representation of the grammar.

- 3. The ellipsis are simple and natural. After the results of a query are displayed, the user can continue to modify the query by entering "and <more-conditions>" or if the user wants all new conditions with the same relations then "where <new-conditions>" is entered. This feature eliminates a lot of un-necessary typing.

- 4. The system is completely modular, which is a result of the design and the independent nature of PROLOG clauses. New or different semantics can be written without making any changes to the parser, or new linguistic features can be added without changing the existing system. The compiler functions completely independent of the database. When a new database is loaded the compiler operates without modification.

- 5. Since the entire system is written in PROLOG, it is easily portable to other machines.

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The original objective of developing a natural language interface to relational data has been successfully implemented. The project required much research in the areas of: artificial intelligence, natural language, parsing and compiling techniques, database, logic programming, and human factors. The author is well aware that some issues in each or the areas have not been addressed, nor could they be in a thesis with such a broad scope. The achieved objective was to define and implement a system that would allow orthogonal, independent development of the database and linguistic sub-components. The emphasis here has been on the following :(1) defining a semantic grammar kernel,(2) defining features to support human-factors considerations, and (3) keeping linguistic and database elements and components independent to provide for modular extentions to the system, to the grammar, and to the database in later projects.

On the basis of the design decisions for this project, the research, and using PROLOG for system implementation, the following conclusions were made:

- PROLOG: The basic features of PROLOG make it especially suitable for applications in high-level language and expert system programming, natural language processing, compiler writing, and database work. The author feels that logic programming, with PROLOG and its future improvements, will become the language of choice for these AI application areas. The Fifth Generation Project should be followed very closely. Much good will come from the effort, even if only the first goal of a

very fast parallel-architecture logic machine is realized.

- Linguistics: For a project of this size, the semantic grammar approach works quite well, and is a good example of the breadth of linguistic complexity that can be achieved with a few rules and simple constructs. Future projects should compare different grammar approaches as well as using the current system as a kernel and layering more sophisticated linguistic features.

- Human Factors: The overriding human factor in this project has been simplicity. With a clean simple initial implementation of a system, it is always possible to extend or modify later. The query language is simple and interactive, providing the computational power of selects and joins and always displays everything that can be infered from a query. There is no quantifier confusion, because the system always assumes 'all' when answering a query. And, the two forms of ellipsis are easy and very natural. One form provides for the continued concatenation of new conditions, while the other form provides for a whole new set of conditions given the same relational context.

- System Design: Both the system design and the independent nature of PROLOG clauses contribute to the modularity and extensibility of this project. The independence encourages parallel development of component parts. The database elements can take full advantage of normalization or other data organization techniques. And, the linguistic or compiler elements can experiment with new grammars for greater linguistic coverage, without concern for the impact on the other system elements.

5.2 Recommendations

The following improvements and modifications are presented here for consideration as possible areas of research for follow-on projects.

- While the code for this system could be transported,at the source code level, from the AVSAIL DEC-10 to the AFIT VAX; some re-coding should be done to treat the intermediate PROLOG structures, in the compiler, as

strings instead of directly as objects. This approach should generalize and simplify the other semantic routines.

- The 'rlx' routine which 'keys' the relations together for a given context needs to be generalized and implemented, with a look-up table, to provide for modular implementation of this function. Then all possible combinations of relations and their inter-relational keys can be understood by the compiler by simply supplying the 'relational-key' table for each database.

- The equivalent of the relational algebra 'project' should be included in the report generation or output routines.

- The semantics and syntax of WH questions needs to be investigated and included in the present grammar.

- An interactive help and error checking routine, which includes a spelling corrector, should be implemented.

- Abstracting all or parts of a query with user defined aliases would be powerful yet easy feature to implement. Similarly the system could learn new ellipsis forms and synonyms from the user.

- Automatic database design tools, could be implemented, that can infer from a well chosen set of queries, how the data should be partitioned, normalized, and organized.

This system can now be used in a modular, extentensible manner to extend research in the areas of logic programming, natural language understanding, knowledge representation, database design and specification, or for human factors studies in man-machine dialogues.

I. Definite Clause Grammars

Prolog's grammar rules provide a convenient notation for expressing definite clause grammars [Colmerauer 1975] [Pereira & Warren 1978]. Definite clause grammars are an extension of the well-known context-free grammars.

A grammar rule takes the general form:-

LHS --> RHS.

meaning "a possible form for LHS is RHS". Both RHS and LHS are sequences of one or more items linked by the standard Prolog conjunction operator ',' .

Definite clause grammars extend context-free grammars in the following ways:-

(1) A non-terminal symbol may be any Prolog term (other than a variable or integer).

(2) A terminal symbol may be any Prolog term. To distinguish terminals from non-terminals, a sequence of one or more terminal symbols is written within a grammar rule as a Prolog list. An empty sequence is written as the empty list '[]'. If the terminal symbols are ASCII character codes, such lists can be written (as elsewhere) as strings. An empty sequence is written as the empty list '[]' or '""'.

(3) Extra conditions, in the form of Prolog procedure calls, may be included in the right-hand side of a grammar rule. Such

procedure calls are written enclosed in '{' '}' brackets.

(4) The left-hand side of a grammar rule consists of a non-terminal, optionally followed by a sequence of terminals (again written as a Prolog list).

(5) Alternatives may be stated explicitly in the right-hand side of a grammar rule, using the disjunction operator ';' as in Prolog.

(6) The cut symbol may be included in the right-hand side of a grammar rule, as in a Prolog clause. The cut symbol does not need to be enclosed in '{' '}' brackets.

As an example, here is a simple grammar which parses an arithmetic expression (made up of digits and operators) and computes its value:-

```
expr(Z) --> term(X), "*", expr(Y), {Z is X * Y}.
expr(Z) --> term(X), "/", expr(Y), {Z is X / Y}.
expr(X) --> term(X).

term(Z) --> number(X), "+", term(Y), {Z is X + Y}.
term(Z) --> number(X), "-", term(Y), {Z is X - Y}.
term(Z) --> number(Z).

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {"0"=<C, C=<"9", X is C - "0"}.
```

In the last rule, C is the ASCII code of some digit.

The question:-

```
?- expr(Z,"-2+3*5+1",[])
```

will compute Z=14. The two extra arguments are explained below.

Now, in fact, grammar rules are merely a convenient "syntactic sugar" for ordinary Prolog clauses. Each grammar rule takes an input string, analyses some initial portion, and produces the remaining portion (possibly enlarged) as output for further analysis. The arguments required for the input and output strings are not written explicitly in a grammar rule, but the syntax implicitly defines them. We now show how to translate grammar rules into ordinary clauses by making explicit the extra arguments.

A rule such as:-

```
p(X) --> q(X).
```

translates into:-

```
p(X,S0,S) :- q(X,S0,S).
```

If there is more than one non-terminal on the right-hand side, as in:-

```
p(X,Y) --> q(X), r(X,Y), s(Y).
```

then corresponding input and output arguments are identified, as in:-

```
p(X,Y,S0,S) :- q(X,S0,S1), r(X,Y,S1,S2), r(Y,S2,S).
```

Terminals are translated using the predicate 'c(S1,X,S2)', read

as "point S1 is connected by terminal X to point S2", and defined by the single clause:-

```
c([X,..S],X,S).
```

Then, for instance:-

```
p(X) --> [go,to], q(X), [stop].
```

is translated by:-

```
p(X,S0,S)  :-  c(S0,go,S1),  c(S1,to,S2),  q(X,S2,S3),
c(S3,stop,S).
```

Extra conditions expressed as explicit procedure calls naturally translate as themselves, eg.

```
p(X) --> [X], {integer(X), X>0}, q(X).
```

translates to:-

```
p(X,S0,S) :- c(S0,X,S1), integer(X), X>0, q(X,S1,S).
```

Similarly, a cut is translated literally.

Terminals on the left-hand side of a rule translate into an explicit list in the output argument of the main non-terminal, eg.

```
is(N), [not] --> [aint].
```

becomes:-

```
is(N,S0,[not,..S]) :- c(S0,aint,S).
```

Disjunction has a fairly obvious translation, eg.

```
args(X,Y) --> dir(X), [to], indir(Y); indir(Y), dir(X).
```

translates to:-

```
args(X,Y,S0,S)  :-  dir(X,S0,S1),  c(S1,to,S2),  indir(Y,S2,S);
indir(Y,S0,S1), dir(X,S1,S).
```

II. Sample of an Interactive Session

```
Prolog-10  version 3.3
Copyright (C) 1981 by D. Warren, F. Pereira and L. Byrd

| ?- [proseq].

sys8 consulted   5472 words      7.22 sec.

yes
| ?- [data].

data consulted   2142 words      3.87 sec.

yes
| ?- talk.
Enter Query or Command
|: Officers where rank is captain or sex is female.

      officer(roger,30,male,captain,married,cs)
      officer(mike,29,male,captain,married,cs)
      officer(linda,23,female,2lt,single,ee)
      officer(mary,33,female,captain,divorced,log)
      officer(mary,33,female,captain,divorced,log)
      officer(susan,24,female,captain,single,log)
      officer(susan,24,female,captain,single,log)
      officer(tom,28,male,captain,married,ee)
      officer(martha,22,female,2lt,single,log)
      officer(kathi,26,female,captain,single,ee)
      officer(kathi,26,female,captain,single,ee)
      officer(nathan,28,male,captain,married,cs)
      officer(james,31,male,captain,single,ee)
      officer(diane,32,female,captain,divorced,ee)
      officer(diane,32,female,captain,divorced,ee)
      officer(jean,33,female,captain,married,log)
      officer(jean,33,female,captain,married,log)
      officer(lucille,34,female,major,married,cs)
      officer(ruth,35,female,major,married,cs)
      officer(barbara,37,female,major,single,ee)
      officer(norma,38,female,ltc,divorced,log)
      officer(carol,21,female,2lt,married,cs)
      officer(evelyn,23,female,2lt,single,ee)
      officer(irene,25,female,1lt,divorced,cs)
      officer(julia,26,female,1lt,married,cs)
      officer(anne,27,female,captain,married,ee)
      officer(anne,27,female,captain,married,ee)
      officer(warren,28,male,captain,married,ee)
      officer(dale,29,male,captain,single,log)
      officer(gordon,30,male,captain,single,cs)
      officer(joyce,31,female,major,divorced,ee)
```

```
      officer(melanie,32,female,captain,married,ee)
      officer(melanie,32,female,captain,married,ee)
      officer(alice,34,female,major,married,cs)
Enter Query or Command
|: where rank is major and sex is female.

      officer(lucille,34,female,major,married,cs)
      officer(ruth,35,female,major,married,cs)
      officer(barbara,37,female,major,single,ee)
      officer(joyce,31,female,major,divorced,ee)
      officer(alice,34,female,major,married,cs)
Enter Query or Command
|: and status is married.

      officer(lucille,34,female,major,married,cs)
      officer(ruth,35,female,major,married,cs)
      officer(alice,34,female,major,married,cs)
Enter Query or Command
|: officers and grades where rank is major and class is ma520.

      grade(joyce,ma520,a-)
      officer(joyce,31,female,major,divorced,ee)

      grade(larry,ma520,b)
      officer(larry,33,male,major,married,log)

      grade(alice,ma520,b+)
      officer(alice,34,female,major,married,cs)

      grade(allen,ma520,b-)
      officer(allen,26,male,major,single,log)

      grade(lucille,ma520,b)
      officer(lucille,34,female,major,married,cs)

      grade(ruth,ma520,c)
      officer(ruth,35,female,major,married,cs)

      grade(barbara,ma520,c)
      officer(barbara,37,female,major,single,ee)

Enter Query or Command
|: grades where name is alice.

      grade(alice,ee345,b)
      grade(alice,cs465,b+)
      grade(alice,ma520,b+)
      grade(alice,ee545,a-)
      grade(alice,ma625,a)
      grade(alice,lg676,a-)
Enter Query or Command
|: where name is nathan.
```

```
      grade(nathan,cs360,a-)
      grade(nathan,ma464,a)
      grade(nathan,ee468,b)
      grade(nathan,ma520,b)
      grade(nathan,lg566,a-)
      grade(nathan,ee655,b-)
Enter Query or Command
|: where class is ee468.

      grade(tom,ee468,b)
      grade(allen,ee468,b+)
      grade(martha,ee468,b+)
      grade(kathi,ee468,a)
      grade(jeff,ee468,a-)
      grade(nathan,ee468,b)
      grade(scott,ee468,b)
      grade(michael,ee468,a-)
      grade(james,ee468,b+)
      grade(diane,ee468,a-)
Enter Query or Command
|: officers where age is greater than 37.

      officer(david,45,male,major,single,ee)
      officer(john,48,male,ltc,married,cs)
      officer(norma,38,female,ltc,divorced,log)
      officer(tracy,39,male,ltc,married,cs)
      officer(gregory,40,male,ltc,married,cs)
Enter Query or Command
|: and status is not divorced.

      officer(david,45,male,major,single,ee)
      officer(john,48,male,ltc,married,cs)
      officer(tracy,39,male,ltc,married,cs)
      officer(gregory,40,male,ltc,married,cs)
Enter Query or Command
|: and major is not cs.

      officer(david,45,male,major,single,ee)
Enter Query or Command
|: grades where name is david.

      grade(david,ma355,c+)
      grade(david,ma464,a-)
      grade(david,cs465,d)
      grade(david,lg566,b+)
      grade(david,ee655,b+)
Enter Query or Command
|: and grade is not d.

      grade(david,ma355,c+)
      grade(david,ma464,a-)
```

```
      grade(david,lg566,b+)
      grade(david,ee655,b+)
Enter Query or Command
|: listing(stack).
[listing,stack,.]

stack(stk,[(grade(_1,_2,_3)',_1=david',not(_3=d))]).

Enter Query or Command
|: officers where sex is female and rank is not captain.

      officer(linda,23,female,2lt,single,ee)
      officer(martha,22,female,2lt,single,log)
      officer(lucille,34,female,major,married,cs)
      officer(ruth,35,female,major,married,cs)
      officer(barbara,37,female,major,single,ee)
      officer(norma,38,female,ltc,divorced,log)
      officer(carol,21,female,2lt,married,cs)
      officer(evelyn,23,female,2lt,single,ee)
      officer(irene,25,female,1lt,divorced,cs)
      officer(julia,26,female,1lt,married,cs)
      officer(joyce,31,female,major,divorced,ee)
      officer(alice,34,female,major,married,cs)
Enter Query or Command
|: and major is not log.

      officer(linda,23,female,2lt,single,ee)
      officer(lucille,34,female,major,married,cs)
      officer(ruth,35,female,major,married,cs)
      officer(barbara,37,female,major,single,ee)
      officer(carol,21,female,2lt,married,cs)
      officer(evelyn,23,female,2lt,single,ee)
      officer(irene,25,female,1lt,divorced,cs)
      officer(julia,26,female,1lt,married,cs)
      officer(joyce,31,female,major,divorced,ee)
      officer(alice,34,female,major,married,cs)

Enter Query or Command
|: officers and grades where major is cs and status is married
|: and grade is not a.

      grade(robert,ma355,b-)
      officer(robert,25,male,1lt,married,cs)

      grade(john,ma355,d)
      officer(john,48,male,ltc,married,cs)

      grade(nathan,cs360,a-)
      officer(nathan,28,male,captain,married,cs)

      grade(scott,cs360,c+)
      officer(scott,29,male,2lt,married,cs)
```

```
grade(lucille,cs360,b-)
officer(lucille,34,female,major,married,cs)

grade(ruth,cs360,b)
officer(ruth,35,female,major,married,cs)

grade(tracy,cs360,b+)
officer(tracy,39,male,ltc,married,cs)

grade(gregory,ee345,a-)
officer(gregory,40,male,ltc,married,cs)

grade(carol,ee345,a-)
officer(carol,21,female,2lt,married,cs)

grade(roger,cs465,b+)
officer(roger,30,male,captain,married,cs)

grade(mike,cs465,b-)
officer(mike,29,male,captain,married,cs)

grade(robert,cs465,c-)
officer(robert,25,male,1lt,married,cs)

grade(nathan,ee468,b)
officer(nathan,28,male,captain,married,cs)

grade(scott,ee468,b)
officer(scott,29,male,2lt,married,cs)

grade(ruth,lg470,a-)
officer(ruth,35,female,major,married,cs)

grade(tracy,lg470,b-)
officer(tracy,39,male,ltc,married,cs)

grade(gregory,lg470,c)
officer(gregory,40,male,ltc,married,cs)

grade(carol,lg470,c)
officer(carol,21,female,2lt,married,cs)

grade(julia,lg470,a-)
officer(julia,26,female,1lt,married,cs)

grade(alice,ma520,b+)
officer(alice,34,female,major,married,cs)

grade(mike,ma520,b+)
officer(mike,29,male,captain,married,cs)
```

```
grade(john,ma520,b-)
officer(john,48,male,ltc,married,cs)

grade(nathan,ma520,b)
officer(nathan,28,male,captain,married,cs)

grade(lucille,ma520,b)
officer(lucille,34,female,major,married,cs)

grade(ruth,ma520,c)
officer(ruth,35,female,major,married,cs)

grade(tracy,ma520,b+)
officer(tracy,39,male,ltc,married,cs)

grade(ruth,cs530,b)
officer(ruth,35,female,major,married,cs)

grade(gregory,ee545,a-)
officer(gregory,40,male,ltc,married,cs)

grade(carol,ee545,b+)
officer(carol,21,female,2lt,married,cs)

grade(julia,ee545,c+)
officer(julia,26,female,1lt,married,cs)

grade(alice,ee545,a-)
officer(alice,34,female,major,married,cs)

grade(mike,lg566,a-)
officer(mike,29,male,captain,married,cs)

grade(john,lg566,b+)
officer(john,48,male,ltc,married,cs)

grade(nathan,lg566,a-)
officer(nathan,28,male,captain,married,cs)

grade(lucille,lg566,b)
officer(lucille,34,female,major,married,cs)

grade(scott,ma625,a-)
officer(scott,29,male,2lt,married,cs)

grade(ruth,ma625,b+)
officer(ruth,35,female,major,married,cs)

grade(carol,ma625,b-)
officer(carol,21,female,2lt,married,cs)

grade(robert,ma625,e)
```

```
officer(robert,25,male,1lt,married,cs)

grade(ruth,cs635,a-)
officer(ruth,35,female,major,married,cs)

grade(tracy,cs635,b+)
officer(tracy,39,male,ltc,married,cs)

grade(carol,cs635,b+)
officer(carol,21,female,2lt,married,cs)

grade(julia,cs635,b)
officer(julia,26,female,1lt,married,cs)

grade(robert,ee655,a-)
officer(robert,25,male,1lt,married,cs)

grade(john,ee655,c)
officer(john,48,male,ltc,married,cs)

grade(nathan,ee655,b-)
officer(nathan,28,male,captain,married,cs)

grade(scott,ee655,b)
officer(scott,29,male,2lt,married,cs)

grade(lucille,lg676,b-)
officer(lucille,34,female,major,married,cs)

grade(tracy,lg676,b)
officer(tracy,39,male,ltc,married,cs)

grade(alice,lg676,a-)
officer(alice,34,female,major,married,cs)

Enter Query or Command
|: and class is ee655.

grade(robert,ee655,a-)
officer(robert,25,male,1lt,married,cs)

grade(john,ee655,c)
officer(john,48,male,ltc,married,cs)

grade(nathan,ee655,b-)
officer(nathan,28,male,captain,married,cs)

grade(scott,ee655,b)
officer(scott,29,male,2lt,married,cs)

Enter Query or Command
|: halt.
```

III. A Sample Relational Database

```
/* key relation ::= officer(NAME,AGE,SEX,RANK,STATUS,ED-CODE). */

officer(roger,30,male,captain,married,cs).
officer(mike,29,male,captain,married,cs).
officer(linda,23,female,'2lt',single,ee).
officer(mary,33,female,captain,divorced,log).
officer(robert,25,male,'1lt',married,cs).
officer(david,45,male,major,single,ee).
officer(susan,24,female,captain,single,log).
officer(john,48,male,ltc,married,cs).
officer(tom,28,male,captain,married,ee).
officer(allen,26,male,major,single,log).
officer(martha,22,female,'2lt',single,log).
officer(kathi,26,female,captain,single,ee).
officer(jeff,27,male,'1lt',married,ee).
officer(nathan,28,male,captain,married,cs).
officer(scott,29,male,'2lt',married,cs).
officer(michael,30,male,'1lt',single,cs).
officer(james,31,male,captain,single,ee).
officer(diane,32,female,captain,divorced,ee).
officer(jean,33,female,captain,married,log).
officer(lucille,34,female,major,married,cs).
officer(ruth,35,female,major,married,cs).
officer(william,36,male,major,single,ee).
officer(barbara,37,female,major,single,ee).
officer(norma,38,female,ltc,divorced,log).
officer(tracy,39,male,ltc,married,cs).
officer(gregory,40,male,ltc,married,cs).
officer(carol,21,female,'2lt',married,cs).
officer(keith,22,male,'2lt',single,ee).
officer(evelyn,23,female,'2lt',single,ee).
officer(melvin,24,male,'2lt',single,log).
officer(irene,25,female,'1lt',divorced,cs).
officer(julia,26,female,'1lt',married,cs).
officer(anne,27,female,captain,married,ee).
officer(warren,28,male,captain,married,ee).
officer(dale,29,male,captain,single,log).
officer(gordon,30,male,captain,single,cs).
officer(joyce,31,female,major,divorced,ee).
officer(melanie,32,female,captain,married,ee).
officer(larry,33,male,major,married,log).
officer(alice,34,female,major,married,cs).

instructor(ma355,lawlis).
instructor(cs360,lamount).
instructor(ee345,lamount).
instructor(lg325,beck).
instructor(ma464,lawlis).
instructor(cs465,milne).
```

```
instructor(ee468,hartrum).
instructor(lg470,beck).
instructor(ma520,black).
instructor(cs530,hartrum).
instructor(ee545,milne).
instructor(lg566,reid).
instructor(ma625,black).
instructor(cs635,seward).
instructor(ee655,seward).
instructor(lg676,reid).

grade(roger,ma355,a).
grade(mike,ma355,a).
grade(linda,ma355,'b+').
grade(mary,ma355,'c-').
grade(robert,ma355,'b-').
grade(david,ma355,'c+').
grade(susan,ma355,'b+').
grade(john,ma355,d).
grade(tom,ma355,b).
grade(allen,ma355,'b+').

grade(martha,cs360,a).
grade(kathi,cs360,a).
grade(jeff,cs360,'b+').
grade(nathan,cs360,'a-').
grade(scott,cs360,'c+').
grade(michael,cs360,b).
grade(james,cs360,b).
grade(diane,cs360,'b-').
grade(jean,cs360,a).
grade(lucille,cs360,'b-').
grade(ruth,cs360,b).
grade(william,cs360,c).
grade(barbara,cs360,a).
grade(norma,cs360,b).
grade(tracy,cs360,'b+').

grade(gregory,ee345,'a-').
grade(carol,ee345,'a-').
grade(keith,ee345,a).
grade(evelyn,ee345,c).
grade(melvin,ee345,'b+').
grade(irene,ee345,b).
grade(julia,ee345,b).
grade(anne,ee345,'b-').
grade(warren,ee345,'b-').
grade(dale,ee345,c).
grade(gordon,ee345,c).
grade(joyce,ee345,'b-').
grade(melanie,ee345,'b+').
grade(larry,ee345,b).
```

```
grade(alice,ee345,b).
grade(roger,ee345,b).
grade(linda,ee345,d).
grade(robert,ee345,a).
grade(susan,ee345,'b+').
grade(tom,ee345,'a-').

grade(martha,lg325,e).
grade(jeff,lg325,b).
grade(scott,lg325,b).
grade(james,lg325,'b+').
grade(jean,lg325,'b+').
grade(ruth,lg325,'b-').
grade(barbara,lg325,a).
grade(tracy,lg325,a).
grade(carol,lg325,b).
grade(evelyn,lg325,'b-').

grade(irene,ma464,b).
grade(anne,ma464,'b+').
grade(dale,ma464,'b+').
grade(joyce,ma464,a).
grade(larry,ma464,'b-').
grade(mike,ma464,a).
grade(mary,ma464,a).
grade(david,ma464,'a-').
grade(john,ma464,'a-').
grade(allen,ma464,a).
grade(kathi,ma464,a).
grade(nathan,ma464,a).
grade(michael,ma464,a).
grade(diane,ma464,b).
grade(lucille,ma464,'b-').

grade(william,cs465,a).
grade(norma,cs465,a).
grade(gregory,cs465,'a-').
grade(keith,cs465,'a-').
grade(melvin,cs465,'a-').
grade(julia,cs465,b).
grade(warren,cs465,b).
grade(gordon,cs465,b).
grade(melanie,cs465,'b+').
grade(alice,cs465,'b+').
grade(roger,cs465,'b+').
grade(mike,cs465,'b-').
grade(linda,cs465,c).
grade(mary,cs465,c).
grade(robert,cs465,'c-').
grade(david,cs465,d).
grade(susan,cs465,e).
grade(john,cs465,a).
```

```
grade(tom,ee468,b).
grade(allen,ee468,'b+').
grade(martha,ee468,'b+').
grade(kathi,ee468,a).
grade(jeff,ee468,'a-').
grade(nathan,ee468,b).
grade(scott,ee468,b).
grade(michael,ee468,'a-').
grade(james,ee468,'b+').
grade(diane,ee468,'a-').

grade(jean,lg470,a).
grade(lucille,lg470,a).
grade(ruth,lg470,'a-').
grade(william,lg470,'a-').
grade(barbara,lg470,'b+').
grade(norma,lg470,'b+').
grade(tracy,lg470,'b-').
grade(gregory,lg470,c).
grade(carol,lg470,c).
grade(keith,lg470,d).
grade(evelyn,lg470,e).
grade(melvin,lg470,'b+').
grade(irene,lg470,a).
grade(julia,lg470,'a-').
grade(anne,lg470,a).

grade(warren,ma520,a).
grade(dale,ma520,a).
grade(gordon,ma520,'a-').
grade(joyce,ma520,'a-').
grade(melanie,ma520,a).
grade(larry,ma520,b).
grade(alice,ma520,'b+').
grade(mike,ma520,'b+').
grade(mary,ma520,'b+').
grade(john,ma520,'b-').
grade(allen,ma520,'b-').
grade(kathi,ma520,'b+').
grade(nathan,ma520,b).
grade(lucille,ma520,b).
grade(ruth,ma520,c).
grade(barbara,ma520,c).
grade(tracy,ma520,'b+').
grade(melvin,ma520,'a-').

grade(scott,cs530,a).
grade(michael,cs530,'a-').
grade(james,cs530,a).
grade(diane,cs530,'a-').
grade(jean,cs530,a).
```

```
grade(lucille,cs530,a).
grade(ruth,cs530,b).
grade(william,cs530,'b+').
grade(barbara,cs530,b).
grade(norma,cs530,'b+').

grade(tracy,ee545,a).
grade(gregory,ee545,'a-').
grade(carol,ee545,'b+').
grade(keith,ee545,b).
grade(evelyn,ee545,'b-').
grade(melvin,ee545,'c+').
grade(julia,ee545,'c+').
grade(anne,ee545,'c-').
grade(warren,ee545,c).
grade(dale,ee545,d).
grade(gordon,ee545,'b+').
grade(joyce,ee545,'a-').
grade(melanie,ee545,e).
grade(larry,ee545,e).
grade(alice,ee545,'a-').

grade(roger,lg566,a).
grade(mike,lg566,'a-').
grade(mary,lg566,'a-').
grade(david,lg566,'b+').
grade(john,lg566,'b+').
grade(allen,lg566,'b-').
grade(kathi,lg566,'b+').
grade(nathan,lg566,'a-').
grade(michael,lg566,'a-').
grade(diane,lg566,'b+').
grade(lucille,lg566,b).
grade(william,lg566,b).
grade(norma,lg566,a).
grade(gregory,lg566,a).
grade(keith,lg566,'a-').
grade(melvin,lg566,'a-').
grade(julia,lg566,a).
grade(warren,lg566,a).
grade(gordon,lg566,'b+').
grade(melanie,lg566,b).

grade(alice,ma625,a).
grade(linda,ma625,'a-').
grade(tom,ma625,'a-').
grade(scott,ma625,'a-').
grade(ruth,ma625,'b+').
grade(carol,ma625,'b-').
grade(anne,ma625,'b+').
grade(larry,ma625,d).
grade(robert,ma625,e).
```

```
grade(susan,ma625,'a-').

grade(martha,cs635,a).
grade(jeff,cs635,a).
grade(james,cs635,a).
grade(jean,cs635,'a-').
grade(ruth,cs635,'a-').
grade(barbara,cs635,'a-').
grade(tracy,cs635,'b+').
grade(carol,cs635,'b+').
grade(evelyn,cs635,'b+').
grade(irene,cs635,b).
grade(julia,cs635,b).
grade(anne,cs635,b).
grade(dale,cs635,'c+').
grade(joyce,cs635,'c+').
grade(larry,cs635,c).

grade(roger,ee655,a).
grade(mike,ee655,a).
grade(linda,ee655,'a-').
grade(mary,ee655,'a-').
grade(robert,ee655,'a-').
grade(david,ee655,'b+').
grade(susan,ee655,'a-').
grade(john,ee655,c).
grade(tom,ee655,c).
grade(allen,ee655,'b+').
grade(martha,ee655,'b+').
grade(kathi,ee655,'b-').
grade(jeff,ee655,'b-').
grade(nathan,ee655,'b-').
grade(scott,ee655,b).
grade(michael,ee655,b).
grade(james,ee655,'a-').

grade(lucille,lg676,'b-').
grade(william,lg676,'b-').
grade(tracy,lg676,b).
grade(keith,lg676,b).
grade(irene,lg676,'b+').
grade(warren,lg676,'b+').
grade(joyce,lg676,'a-').
grade(alice,lg676,'a-').
grade(roger,lg676,a).
grade(linda,lg676,a).


)
```

```
  )


/* system 9 -- with multiple relations and conditions */
/* user access to PROLOG, use of both 'and' and 'or'  */
/* elipsis;  and <conds> .....  where <conds> ......  */

talk :-  repeat,
         write('Enter Query or Command'),nl,
         read_in(Sentence), write(Sentence),nl,
         quecomp(_,Sentence,[]),
         Sentence = [stop|_].
/*****************************************************************/


/* Read in a sentence */

read_in([W|Ws]) :- get0(C), readword(C,W,C1), restsent(W,C1,Ws).

/* given a word and the character after it, read in the
    rest of the sentence */

restsent(W,_,[]) :- lastword(W), !.
restsent(W,C,[W1|Ws]) :- readword(C,W1,C1), restsent(W1,C1,Ws).

/* read in a single word, given an initial character, and
   remembering what character came after the word  */

readword(C,W,C1) :- single_character(C),!, name(W,[C]), get0(C1).
readword(C,W,C2) :- in_word(C,NewC), !,
                    get0(C1),
                    restword(C1,Cs,C2),
                    name(W,[NewC|Cs]).

readword(C,W,C2) :- get0(C1), readword(C1,W,C2).

restword(C,[NewC|Cs],C2) :- in_word(C,NewC), !,
                            get0(C1),
                            restword(C1,Cs,C2).
restword(C,[],C).

/* these characters form words on their own */
single_character(44).  /* , */
single_character(59).  /* ; */
single_character(58).  /* : */
single_character(63).  /* ? */
single_character(33).  /* ! */
single_character(46).  /* . */

/* these characters can appear within a word */
```

```
/* the second in_word clause converts characters to lowercase */

in_word(C,C) :- C>96, C<123.                /* a...z */
in_word(C,L) :- C>64, C<91, L is C+32.     /* A...Z */
in_word(C,C) :- C>47, C<58.                /* 1...9 */
in_word(39,39).                            /*  '    */
in_word(45,45).                            /*  -    */
in_word(61,61).
in_word(43,43).

/* these words terminate a sentence */

lastword('.').
lastword('!').
lastword('?').

/*****************************************************************/

quecomp(Cmd) --> command(Cmd),{write(Cmd),nl}.
command(stop) --> [stop,.].
command(Cmd) --> [Cmd,.],          {call(Cmd)}  .
command(Cmd) --> [Cmd,Parm,.],     {Z =.. [Cmd,Parm],call(Z)}  .


quecomp(Ans) --> [and], aconds, [.],!, {finishup}.
aconds --> cond(F,R,O,V), {mcsemantics(F,R,O,V)}.

quecomp(Ans) --> [where], {clean}, conds, [.],!, {finishup}.

quecomp(Ans)  --> relations, [where], conds, [.],!, {finishup}.

/*****************************************************************/
relations --> {initialize}, rels, {rlxmatch}.

rels --> rel(R), {push(stk,R)},
                 {write(one-rl),nl},
                 {functor(R,F,N)},
                 {pop(count,Cnt)},
                 {Ct is Cnt + 1},
                 {push(count,Ct)},
                 {asserta(position(F,Ct))}.

rels --> rel(R), [and], rels, {pop(stk,Rel)},
                              {write(two-rels),nl},
                              {push(stk,(Rel,R))},
                              {functor(R,F,N)},
                              {pop(count,Cnt)},
                              {Ct is Cnt + 1},
                              {push(count,Ct)},
                              {asserta(position(F,Ct))}.

/*****************************************************************/
```

```
/** this section and the attr section are the only components **/
/** need to be changed when different sets of data are used   **/
/*****************************************************************/
rel(officer(Name,Age,Sex,Rank,Mst,Code)) --> [officers].
rel(officer(Name,Age,Sex,Rank,Mst,Code)) --> [officer].
rel(officer(Name,Age,Sex,Rank,Mst,Code)) --> [students].
rel(officer(Name,Age,Sex,Rank,Mst,Code)) --> [student].
rel(grade(Name,Class,Grade)) -->  [grades].
rel(grade(Name,Class,Grade)) -->  [classes].
rel(grade(Name,Class,Grade)) -->  [grade].
rel(instructor(Class,Iname))  -->  [teachers].
rel(instructor(Class,Iname))  -->  [instructors].
rel(instructor(Class,Iname))  -->  [teacher].
/*****************************************************************/

conds --> cond(F,R,O,V),{scsemantics(F,R,O,V)}.
conds --> cond(F,R,O,V), [and], conds, {mcsemantics(F,R,O,V)}.
conds --> cond(F,R,O,V), [or], conds, {mcorsemantics(F,R,O,V)}.

cond(F,R,O,V) --> attr(F,R),opr(O),vocab(V).
/*****************************************************************/
/**  this is the other interchangeable section for new data   **/
/*****************************************************************/
attr(1,officer) --> [name].
attr(2,officer) --> [age].
attr(3,officer) --> [sex].
attr(4,officer) --> [rank].
attr(5,officer) --> [status].
attr(6,officer) --> [major].

attr(1,grade) --> [name].
attr(2,grade) --> [class].
attr(3,grade) --> [grade].

attr(1,instructor) --> [class].
attr(2,instructor) --> [teacher].
/*****************************************************************/

opr(=) --> [is].
opr(=) --> [equals].
opr(=) --> [=].
opr(~) --> [is,not].
opr(~) --> [~].
opr(>) --> [is,greater,than].
opr(>) --> [>].
opr(<) --> [is,less,than].
opr(<) --> [<].
/*****************************************************************/
vocab(Word) --> [Word].
punc -->  [.].
/*****************************************************************/
rlxmatch :- findall((Rel,Pos),position(Rel,Pos),List),
```

```
            length(List,Lngth),
            asserta(rlslength(Lngth)),
            ( Lngth = 1 -> xmatch1 ;
              Lngth = 2 -> xmatch2 ;
              Lngth = 3 -> xmatch3 ).
xmatch1 :- true.
xmatch2 :- pop(stk,(Rel1,Rel2)),
               position(R,1),
               (R = 'instructor' -> (arg(1,Rel1,Z),
                                     arg(2,Rel2,Z));

                                    (arg(1,Rel1,Z),
                                     arg(1,Rel2,Z))),
           push(stk,(Rel1,Rel2)).
xmatch3 :- pop(stk,((Rel1,Rel2),Rel3)),
           arg(1,Rel1,Z),
           arg(2,Rel2,Z),
           arg(1,Rel2,X),
           arg(1,Rel3,X),
           push(stk,(Rel1,Rel2,Rel3)).

/*****************************************************************/
scsemantics(F,R,O,V) :- rlslength(Lngth),
                (Lngth = 1 -> sc1rsemantics(F,R,O,V);
                 Lngth = 2 -> sc2rsemantics(F,R,O,V);
                 Lngth = 3 -> sc3rsemantics(F,R,O,V)).
sc1rsemantics(F,R,O,V) :-
          pop(stk,B),
          write(sc1r),nl,
          arg(F,B,Q),
          (O == '=' -> push(stk,(B,Q=V)) ;
           O == '~' -> push(stk,(B,not(Q=V))) ;
           O == '>' -> push(stk,(B,Q>V)) ;
           O == '<' -> push(stk,(B,Q<V))).
sc2rsemantics(F,R,O,V) :-
          pop(stk,(One,Two)),
          write(sc2r),nl,
               position(R,P),
               (P = 1 -> arg(F,One,Q) ;
                P = 2 -> arg(F,Two,Q)),
          (O == '=' -> push(stk,(One,Two,Q=V)) ;
           O == '~' -> push(stk,(One,Two,not(Q=V))) ;
           O == '>' -> push(stk,(One,Two,Q>V)) ;
           O == '<' -> push(stk,(One,Two,Q<V))).
sc3rsemantics(F,R,O,V) :-
          pop(stk,(One,Two,Three)),
          write(sc3r),nl,
               position(R,P),
               (P = 1 -> arg(F,One,Q) ;
                P = 2 -> arg(F,Two,Q) ;
                P = 3 -> arg(F,Three,Q)),
          (O == '=' -> push(stk,(One,Two,Three,Q=V)) ;
```

```
            0 == '~' -> push(stk,(One,Two,Three,not(Q=V))) ;
            0 == '>' -> push(stk,(One,Two,Three,Q>V)) ;
            0 == '<' -> push(stk,(One,Two,Three,Q<V))).
/*****************************************************************/

mcsemantics(F,R,O,V) :- rlslength(Lngth),
                (Lngth = 1 -> mc1rsemantics(F,R,O,V);
                 Lngth = 2 -> mc2rsemantics(F,R,O,V);
                 Lngth = 3 -> mc3rsemantics(F,R,O,V)).
mc1rsemantics(F,R,O,V) :-
           pop(stk,(B,Cds)),
           write(mc1r),nl,
           arg(F,B,Q),
           (0 == '=' -> push(stk,(B,Cds,Q=V)) ;
            0 == '~' -> push(stk,(B,Cds,not(Q=V))) ;
            0 == '>' -> push(stk,(B,Cds,Q>V)) ;
            0 == '<' -> push(stk,(B,Cds,Q<V))).
mc2rsemantics(F,R,O,V) :-
           pop(stk,(One,Two,Cds)),
           write(mc2r),nl,
                 position(R,P),
                 (P = 1 -> arg(F,One,Q) ;
                  P = 2 -> arg(F,Two,Q)),
           (0 == '=' -> push(stk,(One,Two,Cds,Q=V)) ;
            0 == '~' -> push(stk,(One,Two,Cds,not(Q=V))) ;
            0 == '>' -> push(stk,(One,Two,Cds,Q>V)) ;
            0 == '<' -> push(stk,(One,Two,Cds,Q<V))).
mc3rsemantics(F,R,O,V) :-
           pop(stk,(One,Two,Three,Cds)),
           write(mc3r),nl,
                 position(R,P),
                 (P = 1 -> arg(F,One,Q) ;
                  P = 2 -> arg(F,Two,Q) ;
                  P = 3 -> arg(F,Three,Q)),
           (0 == '=' -> push(stk,(One,Two,Three,Cds,Q=V)) ;
            0 == '~' -> push(stk,(One,Two,Three,Cds,not(Q=V))) ;
            0 == '>' -> push(stk,(One,Two,Three,Cds,Q>V)) ;
            0 == '<' -> push(stk,(One,Two,Three,Cds,Q<V))).
/*****************************************************************/

mcorsemantics(F,R,O,V) :- rlslength(Lngth),
                (Lngth = 1 -> mc1rorsemantics(F,R,O,V);
                 Lngth = 2 -> mc2rorsemantics(F,R,O,V);
                 Lngth = 3 -> mc3rorsemantics(F,R,O,V)).
mc1rorsemantics(F,R,O,V) :-
           pop(stk,(B,Cds)),
           write(mc1r),nl,
           arg(F,B,Q),
           (0 == '=' -> push(stk,(B,Cds';'Q=V)) ;
            0 == '~' -> push(stk,(B,Cds';'not(Q=V))) ;
            0 == '>' -> push(stk,(B,Cds';'Q>V)) ;
            0 == '<' -> push(stk,(B,Cds';'Q<V))).
```

```
mc2rorsemantics(F,R,O,V) :-
          pop(stk,(One,Two,Cds)),
          write(mc2r),nl,
                position(R,P),
                (P = 1 -> arg(F,One,Q) ;
                 P = 2 -> arg(F,Two,Q)),
          (O == '=' -> push(stk,(One,Two,Cds';'Q=V)) ;
           O == '~' -> push(stk,(One,Two,Cds';'not(Q=V))) ;
           O == '>' -> push(stk,(One,Two,Cds';'Q>V)) ;
           O == '<' -> push(stk,(One,Two,Cds';'Q<V))).
mc3rorsemantics(F,R,O,V) :-
          pop(stk,(One,Two,Three,Cds)),
          write(mc3r),nl,
                position(R,P),
                (P = 1 -> arg(F,One,Q) ;
                 P = 2 -> arg(F,Two,Q) ;
                 P = 3 -> arg(F,Three,Q)),
          (O == '=' -> push(stk,(One,Two,Three,Cds';'Q=V)) ;
           O == '~' -> push(stk,(One,Two,Three,Cds';'not(Q=V))) ;
           O == '>' -> push(stk,(One,Two,Three,Cds';'Q>V)) ;
           O == '<' -> push(stk,(One,Two,Three,Cds';'Q<V))).
/*****************************************************************/
                .
clean  :- rlslength(Lng),
                (Lng = 1 -> (pop(stk,(R1,Cds)),
                             push(stk,R1))          ;
                 Lng = 2 -> (pop(stk,(R1,R2,Cds)),
                             push(stk,(R1,R2)))     ;
                 Lng = 3 -> (pop(stk,(R1,R2,R3,Cds)),
                             push(stk,(R1,R2,R3))) ).

finishup :-
  rlslength(Lng),
        (Lng = 1 -> ( (pop(stk,Done);
                              (pop(stk,(Rel,Cfs;Cbs)),
                               push(stk,(Rel,(Cfs;Cbs))),
                               pop(stk,Done)  )),
                      push(stk,Done), write(Done),nl,
                      changeform(Done,(Rel,(Cds))),
                      findall(Rel,(Rel,(Cds)),Ans)) ;

         Lng = 2 -> ( (pop(stk,Done);
                              (pop(stk,(Rel,Rel2,Cfs;Cbs)),
                               push(stk,(Rel,Rel2,(Cfs;Cbs))),
                               pop(stk,Done)  )),
                      push(stk,Done),
                      changeform(Done,(Rel1,Rel2,(Cds))),
                      findall((Rel1,Rel2),
                                (Rel1,Rel2,(Cds)),Ans));

         Lng = 3 -> ( (pop(stk,Done);
                              (pop(stk,(Rel,R2,R3,Cfs;Cbs)),
```

```
                                    push(stk,(Rel,R2,R3,(Cfs;Cbs))),
                                    pop(stk,Done)  )),
                            push(stk,Done),
                            changeform(Done,(Rel1,Rel2,Rel3,(Cds))),
                            findall((Rel1,Rel2,Rel3),
                            (Rel1,Rel2,Rel3,(Cds)),Ans))),
    write(done),nl,!,
    flatten(Ans,Answer),
    printthe(Answer).
/*****************************************************************/

set(Name,Value) :-  retractall(variable(Name,_)),
                    assert(variable(Name,Value)).

add1(Name,NewValue) :-  retract(variable(Name,Value)),
                        NewValue is Value + 1,
                        assert(variable(Name,NewValue)).

stack_init(Name) :- retractall(stack(Name,_)),
                    assert(stack(Name,[])).

push(Name,Element) :- retract(stack(Name,List)),
                      assert(stack(Name,[Element|List])), !.

pop(Name,Element) :- retract(stack(Name,[Element|List])),
                     assert(stack(Name,List)), !.

retractall(X) :- retract(X), fail.
retractall(X) :- retract((X:-Y)), fail.
retractall(_).

changeform(A,A) :- true.


flatten(A,A) :- var(A), !.
flatten((A,B),C) :- !,
        flatten1(A,C,R),
        flatten(B,R).
flatten(A,A).

flatten1(A,(A,R),R) :- var(A), !.
flatten1((A,B),C,R) :- !,
        flatten1(A,C,R1),
        flatten1(B,R1,R).
flatten1(A,(A,R),R).
/*****************************************************************/

findall(X,G,_) :-
        asserta(found(mark)),
        call(G),
        asserta(found(X)),
        fail.
```

```
findall(_,_,L) :- collect_found([],M), !, L=M.

collect_found(S,L) :- getnext(X), !, collect_found([X|S],L).
collect_found(L,L).

getnext(X) :-  retract(found(X)), !, X \== mark.

printthe([]).
printthe([First|Rest]) :-
        tab(6),
        rlslength(Lng),
           (Lng = 1 -> (write(First),nl) ;
            Lng = 2 -> (changeform(First,(One,Two)),
                        write(One),nl,tab(6),write(Two),nl,nl);
            Lng = 3 -> (changeform(First,(One,Two,Three)),
                        write(One),nl,tab(6),
                        write(Two),nl,tab(6),
                        write(Three),nl,nl)),
        printthe(Rest),!.


initialize :- retractall(position(F,Ct)),
              stack_init(stk),
              stack_init(count),
              push(count,0),!.

not(P) :- call(P), !, fail.
not(P).

/* the end */
```

REFERENCES

[1]
Astrahan, M. M. and Chamberlain D.D.
Implementation of a Structured English Query Language.
ACM 18(10), October, 1975.

[2]
Barr, Avron and Feigenbaum, Edward A.
The Handbook of Artificial Intelligence.
HeurisTech Press, 1981.

[3]
Clark, K.L. and McCabe, F.G.
IC-PROLOG: Aspects of its Implementation.
In Proceedings of the Logic Program Workshop. , July, 1980.
Debrecen, Hungary.

[4]
Clark, K. L. and Tarnlund, S. A. (Ed.)
LOGIC PROGRAMMING.
Academic Press, 1982.

[5]
Clocksin, W. F. and Mellish, C. S. .
Programming in Prolog.
Springer-Verlag, 1981.

[6]
Codd, E. F.
A Relational Model for Data for Large Shared Data Banks.
CACM 13(6), June, 1970.

[7]
Codd, E. F.
Relational Database: A Practical Foundation for Productivity.
CACM 25(2), February, 1982.

[8]
Colmerauer, A.
Metamorphis Grammars.
In Goos, G. and Hartmanis, J., editors, Lecture Notes in Computer Science, . Springer-Verlag, 1979.

[9]
Dahl, Veronica.
Translating Spanish into Logic through Logic.
American Journal of Computational Linguistics 7(3), July-September, 1981.

[10]
Dahl, Veronica .
On Database Systems Development Through Logic.
ACM Transactions on Database Systems 7(1), March, 1982.

[11]
Dahl, V. and Webber, B.
Knowledge Representation.
IEEE-COMPUTER 16(10), October, 1983.

[12]
Date, C. J.
An Introduction to Database Systems, Third Edition.
Addison-Wesley, 1981.

[13]
Ehrenreich, S. L.
Deign Recommendations for Query Languages.
Technical Report TR 484; AD-A115894, U.S. Army Research Institute for the Behavioral and Social Sciences, September, 1980.

[14]
Eisinger, Norbert and Kasif, S. and Minker J.
Logic Programming: A Parallel Approach.
Technical Report TR-1124, University of Maryland, December, 1981.

[15]
Feigenbaum, Edward and McCorduck, Pamela.
Land of the Rising Fifth Generation Computer.
HIGH TECHNOLOGY , June, 1983.

[16]
Feigenbaum, Edward and McCorduck, Pamela.
The Fifth Generation.
Addison-Wesely, 1983.

[17]
Gazdar, Gerald.
Phrase Structure Grammars and Natural Language.
In Proceedings of the 1983 IJCAI. International Joint Conf. on Artificial Intelligence (IJCAI), August, 1983.

[18]
Gevarter, William B.
An Overview of Computer-Based Natural Language Processing.
Technical Report Technical Memorandum 85635, NASA, April, 1983.

[19]
Hendrix, G. G. and Sacerdoti, E. D.
Natural-Language Processing: The Field in Perspective.
Byte Magazine , September, 1981.

[20]
Kaplan, S. J. et al.
Special Section on Natural Language Systems.
ACM-SIGART (79), January, 1982.
Includes over 150 pages of references to NL work though out the world.

[21]
Rosenblatt, Roger
Machine of the Year.
TIME 121(1), January 3rd, 1983.

[22]
Kim, Won.
Relational Database Systems.
Computing Surveys (ACM) 11(3), September, 1979.

[23]
Robert Kowalski.
Algorithm = Logic + Control.
CACM 22(7), July, 1979.

[24]
Robert Kowalski.
Logic for Problem Solving.
Elsevier North Holland, New York, 1979.

[25]
Manna, Zohar.
Mathematical Theory of Computation.
McGraw-Hill, 1974.

[26]
Michael C. McCord.
Using Slots and Modifiers in Logic Grammars for Natural Language.
Artificial Intelligence 18(3), May, 1982.

[27]

Pereira, Luis M. and Pereira, Fernando and Warren, D.H.D.
USER'S GUIDE to DECsystem-10 PROLOG
Dept. of Artificial Intelligence, University of Edinburgh, 1978.

[28]

Pereira, Fernando and Warren, David H. D.
Definite Clause Grammars Compared with Augmented Transition Networks.
Research Paper 116, University of Edinburgh, Dept. of Artificial Intelligence, 1979.

[29]

Robinson, J. A.
A Machine-oriented Logic Based on the Resolution Principle.
J. ACM 12(1), January, 1965.

[30]

Schank, R. C. and Abelson, R. P.
Scripts, Plans, Goals and Understanding.
Lawerance Erlbaum, Hillsdale, N.J., 1977.

[31]

Shapiro, Ehud Y.
The Fifth Generation Project - A Trip Report.
CACM 26(9), September, 1983.

[32]

Shapiro, Ehud Y. .
Algorithmic Program Debugging.
MIT Press, 1983.
An ACM Distinguished Dissertation-1982.

[33]

Thomas, J. C.
Quantifiers and Question-asking.
Technical Report RC-5866, IBM Thomas J. Watson Reaearch Center, February, 1976.

[34]

Treleaven, Philip and Lima, Isabel G.
Japan's Fifth-Generation Computer Systems.
IEEE-COMPUTER , August, 1982.

[35]

Waltz, David L.
Natural Language Interfaces.
ACM-SIGART (61), February, 1977.

[36]
Waltz, David L.
An English Language Question Answering System for a Large Relational Database.
CACM 21(7), July, 1978.

[37]
Warren, David H.
Implementing PROLOG - Compiling Logic Programs.
Technical Report D.A.I. 39 & 40, Univ. of Edinburgh, 1977.

[38]
Warren, David H. D.
Logic Programming and Compiler Writing.
Software - Practice and Experience 10(2), January, 1979.

[39]
Warren, David H. D. .
Efficient Processing Of Interactive Relational Database Queries Expressed In Logic.
In Very Large Data Base. IEEE, 1981.

[40]
Warren, David H. D.
A View of the Fifth Generation and Its Impact.
The AI Magazine 3(4), Fall, 1982.

[41]
Warren, David H. D. and Pereira, Fernando C. N.
An Efficient Easily Adaptable System for Interpreting Natural Language Queries.
American Journal of Computational Linguistics 8(3-4), July-December, 1982.

[42]
Winograd, Terry.
Language as a Cognitive Process.
Addison-Wesley, 1983.

[43]
Yokoi, Toshio , et al.
Logic Programming and a High-Performance Personal Computer.
In T. Moto-oka, editor, FIFTH GENERATION COMPUTER SYSTEMS, North-Holland, 1982.

VITA

Roger P. White was born on 25 April 1951, in Billings Montana. He graduated from Hillcrest High School (1969) in Salt Lake City, and later attended the University of Utah from which he received a Bachelor of Science in Computer Science in June 1978. He is a Distinguished Graduate of the USAF ROTC and entered the Air Force on active duty in October 1978 at Wright-Patterson AFB. He has served assignments in the Acquisition Logistics Division of the Engine SPO and in the Support Software Division of the Avionics Laboratory before entering the Air Force Institute of Technology, in June 1982. He is a member of ACM, SIGART, SIGPLAN, and SIGSOFT.

Permanent address: 2200 East 7110 South
Salt Lake City
Utah, 84121

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/EE/83D-22	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: SCHOOL OF ENGINEERING; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB, OHIO 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: RADC/COA; 8b. OFFICE SYMBOL (If applicable): RADC/COA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Rome Air Development Center Griffiss Air Force Base Rome, New York 13441	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19 (unclassified)	
12. PERSONAL AUTHOR(S): White, Roger P.	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): 1983 December 16; 15. PAGE COUNT: 85
16. SUPPLEMENTARY NOTATION	LYNN E. WOLAVER 7 Feb 84 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Natural language interface, database, PROLOG programming language, fifth generation computing, semantic grammar

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A Natural language interface to a PROLOG database.
A natural language interface to a relational database was developed with careful consideration given to the human factors needed to aid the novice user in accessing data. The entire system is written in DEC-10 PROLOG, with three distinguishing contributions: (1) a simple grammar was developed to parse phrases like: "Officers where rank is Captain and status is single and age is less than 32"; (2) some original techniques were developed to preserve common variable instantiations in complex PROLOG objects, while being constructed by the parser, (3) the human factors that contribute to the system are: a help file to aid user perception of the data, a simple grammar to learn and use, two forms of ellipsis, user-defined aliases, and limited use of quantifiers.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: MILNE, ROBERT CAPTAIN, US ARMY	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576; 22c. OFFICE SYMBOL: AFIT/ENG


FILM
4-8